

ייעוץ מקצועי:
מאיר קלטר



הוצאת הוד-עמי
לספרי מחשבים

Win32API ומבוא ל-MFC
למתכנתי Visual C++ 6

Win32API
ומבוא ל-MFC
למתכנתי
Visual C++ 6

יש להתעלם מכל מה שנכתב על התקליטור.

אין תקליטור מצורף עם תוכנה.

את קוד המקור ניתן להוריד מאתר הוד-עמי בתיקיה "קבצי תרגול לספרים"

עורכת ראשית: **שרה עמיהוד**

עריכה ועיצוב: **רמה שנקלר**

ייעוץ מקצועי: **מאיר קלטר**

עיצוב עטיפה: **ישראל מצגר**

לשם שטף הקריאה כתוב ספר זה בלשון זכר בלבד. ספר זה מיועד לגברים ונשים כאחד ואין בכוונתנו להפלות או לפגוע בציבור המשתמשים/ות.

- **טלפון: 09-9564716**
- **פקס: 09-9571582**
- **דואר אלקטרוני: info@hod-ami.co.il**
- **אתר באינטרנט: www.hod-ami.co.il**

Win32API
ומבוא ל-MFC
למתכנתי
Visual C++ 6

ייעוץ מקצועי:

מאיר קלטר



הוצאת הוד-עמי
לספרי מחשבים



Win32API and introduction to MFC

הוצאת הוד-עמי
לספרי מחשבים בע"מ
ת.ד. 6108 הרצליה 46160
טלפון: 09-9564716 פקס: 09-9571582
info@hod-ami.co.il

הודפס בישראל 2000

HOD-AMI Ltd.
P.O.B. 6108, Herzliya
ISRAEL, 2000

ISBN 965-361-255-7 מסת"ב

תוכן עניינים מקוצר

תוכן עניינים

פרק 7: מפות סיביות, קבצי-על וסמלים (Bitmaps, Metafiles ו-Icons) 160

פרק 8: מבט מקרוב על פקדים 196

פרק 9: הקדמה לפסי גלילה (Scroll Bars) 214

פרק 15: ניהול זיכרון 391

פרק 16: תהליכים ומטלות 420

פרק 17: קלט/פלט בחלונות 481

פרק 1

הקדמה

ספר זה הוא, בראש ובראשונה, מדריך מעשי לתכנות בסביבת חלונות. לפיכך אין הוא עוסק באופן מיוחד בהיבטים התיאורטיים של חלונות, אלא רק כאשר הם נוגעים במישרין לכתיבת תוכניות. במקום זה, הספר מתווה גישה מעשית-התנסותית, שתביא אותך לידי כתיבת יישומי חלונות תוך זמן קצר ביותר.

למרות האמור בפיסקה הקודמת, לפני שתוכל להפוך למתכנת בסביבת חלונות, יהיה עליך להבין במונחים כלליים כיצד פועלת תוכנה זו, על אילו תפיסות תכנון היא מבוססת, וכיצד היא מנהלת את המחשב שלך. חשוב להבין גם במה שונה חלונות מקודמותיה: DOS ו-Windows 3.11. בפרק זה נסקור את חלונות, ונבחן את נקודות הדמיון והשוני.

אם לא כתבת מעולם תוכנית לסביבת חלונות, רוב המידע בספר זה יהיה חדש עבורך. פשוט התאזר בסבלנות. אם תתקדם באופן שיטתי, תיווכח שעד שתגיע לסוף הספר, **תהפוך למתכנת מיומן בסביבת חלונות.**

ונקודה נוספת לסיום: חלונות היא סביבת תכנות מקיפה ומורכבת עד מאוד. לא ניתן למצות אותה במלואה בספר אחד (תיאור ממצה בוודאי יסתכם בכמה וכמה כרכים!). ספר זה דן בכל **מרכיבי התכנות בסביבת חלונות המשותפים לכל התוכניות הנפוצות, ובחידושים החשובים והייחודיים של חלונות**. לאחר שתשלים ספר זה, תרכוש הבנה מספקת בתכנות בסביבת חלונות ותוכל בקלות לחקור כל אחת ממערכות המשנה שלה.

המאפיין החשוב ביותר בחלונות 9x הוא בכך שהיא **מערכת הפעלה של 32 סיביות**, כפי שיתברר לך ככל שתתקדם בלימוד בספר זה. המעבר לסביבת 32 סיביות איפשר לחלונות להותיר מאחור חלק ניכר מהבעיות והמוזרויות הקשורים במערכות הישנות יותר של 16 סיביות.

ריבוי משימות מבוסס על מטלות

כפי שבוודאי ידוע לך, חלונות היא מערכת הפעלה המתוכננת ל**ריבוי משימות** (Multitasking). כפועל יוצא מכך, היא מסוגלת להריץ במקביל שתי תוכניות או יותר - **ריבוי תוכניות** (Multiprogramming). מובן שהתוכניות חולקות ביניהן את ה-CPU ועל כן, מבחינה טכנית הן אינן רצות בו-זמנית; אולם בשל מהירות המחשב, התחושה היא שההרצה אמנם בו-זמנית. חלונות תומכת בשני סוגים של ריבוי משימות: כזה המבוסס על תהליכים, וכזה המבוסס על מטלות. **תהליך** (Process) הוא תוכנית הנמצאת בהרצה. העובדה שחלונות מסוגלת לריבוי משימות כשמדובר בתהליכים, פירושה שהתוכנה מסוגלת להריץ בעת ובעונה אחת שתי תוכניות או יותר. חלונות תומכת, אם כן, בריבוי משימות המבוסס על תהליכים בסגנון הישן, המוכר לך בוודאי.

הסוג השני של ריבוי-משימות בסביבת חלונות מבוסס על מטלות. **מטלה** (Thread) היא יחידה ניתנת **לביצוע** (Dispatch) של קוד להרצה. מקורו של השם ברעיון "מטלות ההרצה". לכל תהליך לפחות מטלה אחת; בחלונות יכול כל תהליך להיות מורכב מכמה מטלות.

מכיון שחלונות מבצעת ריבוי משימות מבוסס על מטלות, וכל תהליך יכול להיות בן כמה מטלות, עשוי להיווצר מצב שבו שני מקטעים, או יותר, של תהליך נתון יוצאים אל הפועל בו-זמנית. למעשה, הנחה זו נכונה. לכן, כשאתה עובד עם חלונות, באפשרותך ליצור ריבוי משימות לגבי תוכניות ולגבי מקטעים שונים של תוכנית מסוימת. כפי שיתברר לך בהמשך ספר זה, תכונה זו מאפשרת לך לכתוב תוכניות יעילות עד מאוד.

ממשק בשיטת הקריאות

אם יש לך רקע ב-DOS, בוודאי ידוע לך שההשקה עם DOS מתבצעת באמצעות **פסיקות תוכנה** (Software Interrupts) שונות. לדוגמה, הפסיקה המקובלת ב-DOS היא 0x21. אולם, למרות שהגישה לשירותי DOS באמצעות פסיקות תוכנה בהחלט מקובלת (נוכח האפשרויות המוגבלות של מערכת ההפעלה DOS), אין זו דרך יעילה כלל וכלל ליצור השקה עם מערכת הפעלה מתוחכמת המסוגלת לריבוי משימות, כמו חלונות. חלונות משתמשת **בממשק בשיטת הקריאות** (Call-Based Interface).

שיטה זו בחלונות פועלת באמצעות סדרה גדולה של פונקציות שהמערכת מגדירה, והן מאפשרות גישה למאפיינים השונים של מערכת ההפעלה. במקובץ, נקראות הפונקציות הללו **ממשק תכנות יישומים**, או בקיצור **API** (Application Programming Interface). API מכיל כמה מאות פונקציות שתוכנית היישומים שלך מפעילה כדי לתקשר עם חלונות. פונקציות אלו כוללות את כל הפעולות הנחוצות הקשורות במערכת ההפעלה, כגון הקצאת זיכרון, פלט למסך, יצירת חלונות, וכדומה.

ספריות בקישור דינמי (DLL)

כיון ש-API מכיל כמה מאות פונקציות, עלול להיווצר אצלך הרושם שכל תוכנית המהודרת לחלונות מקושרת לכמות נכבדה של קוד, דבר היוצר כפילות קוד בכל תוכנית. אולם בפועל אין הדבר כך. במקום זה, הפונקציות הכלולות ב-API של חלונות שמורות **בספריות קישור דינמי**, או בקיצור **DLL** (Dynamic Link Libraries). לכל תוכנית יש גישה לספריות אלה בשעה שהיא רצה. סעיף זה מסביר כיצד פועל הקישור הדינמי.

פונקציות API של חלונות שמורות במתכונת ניתנת להעברה, במסגרת DLL נתון. במהלך שלב ההידור, כאשר התוכנית שלך מפעילה פונקציה באמצעות API, **המקשר** (Linker) אינו מוסיף לגרסת ההרצה של התוכנית שלך את הקוד המתאים לאותה פונקציה. במקום זה, הוא מוסיף לתוכנית הוראות טעינה עבור אותה פונקציה, כגון שם הפונקציה ושם ספריית DLL שבה היא שמורה. כאשר התוכנית רצה, נטענות על ידי **הטוען** (Loader) של חלונות גם שגרות API נחוצות. בדרך זו, אין צורך להכניס בפועל לכל תוכנית את קטעי הקוד של כל פונקציה. פונקציות API מתוספות רק כאשר היישום נטען אל תוך הזיכרון לצורך ההרצה.

שיטת הקישור הדינמי טומנת בחובה כמה יתרונות חשובים. ראשית, כיון שלמעשה כל התוכניות משתמשות בפונקציות API, הקישור הדינמי מונע בזבוז מקום בדיסק, דבר שהיה קורה אילו היה צורך להוסיף בפועל לכל תוכנית להרצה עותקים של קטעי הקוד של פונקציות API המתאימות. שנית, ניתן לבצע שדרוג ולהכניס שיפורים בחלונות 9x בפשטות רבה, על ידי שינוי השגרות של ספריות DLL. באופן כזה, אין צורך להדר מחדש את התוכניות הקיימות של היישומים השונים.

כאשר תשתמש ביישומי חלונות, יתברר לך שכמה מרכיבי בקרה שבים ומופיעים בתדירות גבוהה למדי. ביניהם נמנים סרגל הכלים, **בקרת הסיבוב** (Spin Control), המבט על העץ, ותיבת הסטטוס. בחלונות 9x הוגדרו מספר **פקדים משותפים**, חדשים ומלהיבים, הזמינים לכל היישומים (בהמשך הספר תלמד כיצד להפעיל כמה מהם). השימוש בפקדים החדשים האלה מעניק ליישום שלך את החזות המודרנית, המסמנת אותו בבירור כתוכנית של חלונות 9x.

תורי קלט

תורי קלט (Input Queues) מכילים מסרים, כגון לחיצות על מקשים, או פעילות באמצעות העכבר, עד שניתן לשולח אותן לתוכנית שלך. ב-Windows 3.11 יש רק תור קלט אחד לכל המשימות הרצות בנקודת זמן מסוימת במערכת. בחלונות 9x מוקצה תור קלט לכל מטלה. היתרון הטמון בכך הוא שאף תהליך לא יכול להאט את מהירות הביצוע של המערכת כולה בכך שהוא מגיב באיטיות למסרים שלו.

הגישה של ריבוי תורי קלט היא תוספת חשובה, אך לשינוי זה אין שום השלכה ישירה על הדרך שבה תתכנת עבור חלונות 9x.

מטלות ותהליכים

חלונות תומכת גם בריבוי משימות המבוסס על מטלות. תוכניות ישנות של חלונות ירוצו יפה תחת חלונות 9x בלי שום שינוי, אך ודאי תרצה לשפר אותן כדי שתוכל לנצל את אפשרויות ריבוי משימות המבוסס על מטלות.

פקדים

בעבר, היה השימוש ביישומים **מבוססי טקסט** (כלומר, לא חלונאיים) מתוך חלונות מסורבל למדי. חלונות 9x תומכת בסוג מיוחד של חלונות, הנקראים **פקדים** (Control). חלון מסוג זה מספק לך **סביבת סמן-פקודה** באמצעות ממשק רגיל, מבוסס טקסט. פרט להיותו מבוסס טקסט, פועל הפקד (ומופעל) כמו כל חלון אחר. תוספת זו של פקד מבוסס טקסט מאפשרת הרצה של יישומים לא-חלונאיים בסביבת חלונות מלאה, ומאפשרת ליצור בקלות רבה תוכניות שירות קצרות לשימוש חד-פעמי. חשוב מכך, מרכיב זה של חלונות מסוג פקד בחלונות 9x מהווה למעשה, הכרה בכך שיש היגיון בחלק מהיישומים מבוססי טקסט, ומעתה ניתן לנהלם ולהפעילם כחלק מהסביבה הכוללת של תוכנת חלונות.

מיעון כתובות רציף

קיבולת הזיכרון הווירטואלי העומד לרשות יישומי חלונות הוא 4 GB! יתר על כן, מרחב מיעון הכתובות הזה **רציף** (Flat). שלא כמו ב-Windows 3.11, DOS ויתר מערכות ההפעלה ממשפחת 8086, המשתמשות בזיכרון מקוטע על פי סגמנטים, חלונות 9x מתייחסת לזיכרון כאילו היה רציף. מכיון שהיא משתמשת בשיטת זיכרון וירטואלי, לרשות **כל יישום** עומד זיכרון **בכל כמות** שהוא עשוי להזדקק לה (באופן סביר). המעבר למיעון כתובות רציף שקוף ברובו עבור המתכנת, המסיר מעליו חלק ניכר מהטיפול המייגע והמתסכל במיעון כתובות לפי השיטה הישנה, המקוטעת.

פקדים כלליים חדשים

Windows 3.11 תמכה בכמה פקדים רגילים, כגון לחצנים, תיבות סימון, כפתורי רדיו, תיבות עריכה, וכדומה. חלונות כוללת תמיכה בפקדים הרגילים הללו, אך הוגדרו בה גם כמה פקדים חדשים. הפקדים החדשים נקראים **פקדים כלליים** (Common Controls), וכוללים בין השאר, פריטים כמו סרגלי כלים, tool tips, תיבות סטטוס, תיבות התקדמות, תיבות מעקב ועוד. הפקדים החדשים משפרים את הממשק למשתמש ומעניקים חזות "מודרנית" ליישומים המשתמשים בהם.

הקשר עם NT

בוודאי שמעת על תוכנת Windows NT. ייתכן שאף השתמשת בה. זוהי מערכת ההפעלה המשוכללת ביותר של מיקרוסופט, המבוססת על חלונות. ישנם דברים רבים המשותפים ל-WindowsNT ולחלונות 9x. שתי התוכנות תומכות במיעון כתובות רציף בשיטת - 32 סיביות. שתיהן תומכות בריבוי משימות מבוסס על מטלות, ושתיהן תומכות בממשק המבוסס על **עמדות שליטה**. ואולם אין לבלבל בין שתי התוכנות, שכן אין ביניהן זהות. למשל, Windows NT נוקטת גישה מיוחדת ליישום מערכות הפעלה, גישה המבוססת על דגם **לקוח/שרת**. בחלונות 9x אין זה כך. Windows NT תומכת במערכת אבטחה מלאה; בחלונות אין זה כך.

אין ספק שחלק גדול מהטכנולוגיה הבסיסית שפותחה לצורך יצירת Windows NT שימשה גם לבניית חלונות 9x, אך שתי הגרסאות שונות מאוד זו מזו. עם זאת, רצוי לציין שגירסה 4 של Windows NT קיבלה את חזותה של חלונות 9x, כלומר, ממשק המשתמש דומה מאוד.

עם איזו תוכנה להריץ את התוכניות בספר?

ההידור של קטעי הקוד בספר זה נעשה באמצעות מהדר **Visual C++ 6.0** של מיקרוסופט שנפוץ אצל מתכנתים רבים. סביר להניח שתוכל להדר את התוכניות הכלולות בספר זה באמצעות כל מהדר תואם חלונות. הדוגמאות השונות נכתבו בשפת C/C++ הרגילה, וניתן להדר אותן באמצעות כל מהדר ל- C/C++.

לספר זה מצורפים 2 תקליטורים. באחד מהם תמצא את **מהדר Visual C++ 6 בגירסה מיוחדת** - Introductory Edition של מיקרוסופט.

התוכניות לא הורצו ונבדקו בגירסה זו. התוכנה ניתנת כבונוס ללא תשלום לרוכשי הספר והוצאת הוד-עמי אינה מספקת תמיכה במוצר.

Win2000/Win98

ספר זה מתייחס לגירסה של Visual C++ 6.0 שעבדה עם WINAPI שהתאימו ל-Windows 95. קיימים ServicePacks נוספים שנותנים את האפשרות להשתמש עם תכונות ופקדים שנוספו במערכות ההפעלה הבאות.

כתובת URL טובה מאוד לחיפוש מידע מעודכן היא: Msdn.Microsoft.com

על התקליטורים המצורפים

קרא את קובץ ONCD.DOC בתקליטור כדי לקבל מידע על תכולת התקליטור המצורף.

קרא בהמשך כיצד להעתיק את קבצי המקור הרלוונטים לספר זה.

לספר מצורפים 2 תקליטורים.

באחד מהם תמצא את **המהדר Visual C++ 6 Intoductory Edition בגירסה מיוחדת** של מיקרוסופט. **התוכניות לא הורצו ונבדקו בגירסה זו. התוכנה ניתנת כבונוס ללא תשלום לרוכשי הספר והוצאת הוד-עמי אינה מספקת תמיכה במוצר.**

בתקליטור השני המצורף לספר זה תוכל למצוא מספר דברים:

- ✰ **קטלוג HTML** - קטלוג ספרי המחשבים האינטראקטיבי של **הוצאת הוד-עמי** בו וכל לצפות בספרי ההוצאה ולקרוא פרקים לדוגמה מתוך חלק מהספרים. לשם קריאת הפרקים יש להתקין את תוכנת Adobe Acrobat Reader שנמצאת בתקליטור. הקטלוג מומלץ לצפייה באמצעות Internet Explorer 5 שמצורף בתקליטור.
- ✰ מספר תוכנות עזר שימושיות.
- ✰ קבצי קוד מקור.

✍ **הערה:** אם מנהל התקן כונן התקליטורים המותקן הוא 16 סיביות (וזה יכול להיות גם אם הכונן חדש) - ייתכן שתראה רק את 8 התווים הראשונים של שם הקובץ (במקרה שהמקור ארוך יותר), או ייתכן שתראה שהתיקיות ריקות.

הטוב ביותר הוא להתקין מנהל התקן 32 סיביות, או לקנות כונן תקליטורים חדש ולוודא שמצורף אליו מנהל התקן 32 סיביות.

היכן נמצאים הקבצים הקשורים לספר זה?

התיקיה הרלוונטית לספר זה: 59285

תחת תיקיה **X:\Books\59285** (החלף את האות X באות הכונן המתאימה) תמצא תיקיות משנה: תיקיה עבור כל פרק: תיקיה Chap01 עבור פרק 1, תיקיה Chap02 עבור פרק 2 וכך הלאה. בכל תיקיה נמצאים קטעי קוד המקור לפי הדוגמאות שבספר.

אנו מבקשים להתייחס **רק** לקבצים הנמצאים בתיקיה הרלוונטית לספר תחת Books. בתיקיות אחרות נמצאים קבצים הרלוונטיים לספרים אחרים של ההוצאה.

העתקת קבצי המקור לדיסק

הקבצים נמצאים בתיקיה **X:\books\59285** מסודרים לפי תיקיות - תיקיה עבור כל פרק. כדי להעתיק את הקבצים לדיסק:

1. לחץ על לחצן התחל, תוכניות, סייר Windows.
2. הצג את תוכן התיקיה Books אשר בתקליטור.
3. סמן את התיקיה 59285 וגרור אותה לתיקיה כלשהי בדיסק.

מכיון שמקור הקבצים הוא התקליטור, הם מסומנים לקריאה בלבד. רצוי לשנות מאפיין זה בדרך זו:

1. דרך הסייר היכנס לתיקיה בדיסק, שבה נמצאים הקבצים שהעתקת.
2. סמן קובץ מסוים או את כולם על-ידי Ctrl+A.
3. הצב את סמן העכבר מעל האזור המסומן ולחץ לחיצה ימנית בעכבר.
4. מתפריט הקיצור בחר **מאפיינים**.
5. בטל את הסימון בתיבה **קריאה בלבד** (דאג שתיבה זו תהיה ריקה).
6. לחץ על **החל**, לחץ על **אישור**.

אפשרות אחרת:

1. לחץ על לחצן התחל, תוכניות, הפנייה ל-MS-DOS.
2. בחלון DOS רשום: attrib -r +a c:\59285*.* /s.

אם העתקת את הקבצים לתיקיה אחרת, רשום זאת במקום C:\59285.

פרק **2**

תכנות בחלונות 9x, סקירה

בפרק זה נערוך היכרות עם תכנות בסביבת חלונות 9x. לפרק שתי מטרות עיקריות: ראשית, נגדיר את קשרי הגומלין המחייבים בין מערכת ההפעלה חלונות (Windows) לבין התוכניות הרצות תחתיה, ואת הכללים שחובה על כל יישום של חלונות לציית להם. שנית, נפתח בפרק זה שלד של יישום, אשר ישמש בסיס, או מסגרת, לשאר התוכניות שנפתח בספר זה עבור חלונות. בהמשך תכיר כמה תכונות המשותפות לכל התוכניות של חלונות. המסגרת שתיבנה בפרק זה תכיל את כל אותן תכונות משותפות.

2.1 מבט על התכנות בחלונות 9x

היעד של תוכנת חלונות 9x (וחלונות בכלל) לאפשר לכל מי שיש לו היכרות בסיסית עם המערכת להריץ, בלי הכשרה מוקדמת, כמעט כל יישום שאפשר להעלות על הדעת. כדי להגשים יעד זה, מכילה חלונות ממשק עקבי למשתמש. להלכה, אם אתה יודע כיצד להריץ תוכנית אחת המבוססת על חלונות, תדע להריץ כל תוכנית אחרת. רוב התוכניות השימושיות מחייבות הכשרה כלשהי כדי שניתן יהיה לנצלן באורח יעיל, אבל ההדרכה במקרה שלנו מסתכמת ב**מה עושה התוכנית**, ואין צורך להסתבך בשאלות כמו ***איך על המשתמש לתקשר איתה***. למעשה, חלק ניכר מהקוד ביישומים חלונאיים משמש רק לתמיכה בממשק המשתמש.

לפני שנמשיך, חובה לומר שלא כל תוכנית המסוגלת לרוץ תחת חלונות 9x, תציג בהכרח ממשק בסגנון חלונאי בפני המשתמש. בהחלט ניתן לכתוב תוכניות לחלונות שאינן מנצלות את מרכיבי הממשק החלונאי. כדי ליצור תוכנית בסגנון חלונות, עליך לנקוט כמה פעולות מכוונות, ולהשתמש בטכניקות המתוארות בספר זה. רק התוכניות הנכתבות תוך שימוש באפשרויות הגלומות בחלונות, ייראו ויעניקו תחושה של תוכניות חלונאיות. ניתן בהחלט לוותר על הפילוסופיה העיצובית הבסיסית של חלונות, אך רצוי שתהיה לך סיבה טובה לעשות זאת, כיון שהדבר עשוי לעורר תחושה של אי-נוחות אצל מי שאמור להשתמש בתוכניות שלך. ככלל, רצוי שיישומים שאתה

כותב לחלונות יופעלו באמצעות הממשק הרגיל של חלונות, ויעוצבו לפי עקרונות העיצוב הרגילים של חלונות.

תוכנת חלונות 9x (Windows 9x) היא בעלת אוריינטציה גרפית, כלומר, היא מכילה **ממשק משתמש גרפי** - **GUI** (Graphical User Interface). בשוק נמצא מיגוון רחב של פריטי חומרה לגרפיקה ושיטות וידאו, וחלונות מסוגלת להתמודד עם רוב ההבדלים ביניהם. כלומר, בדרך כלל התוכניות שלך לא צריכות להביא בחשבון את סוג החומרה, או את שיטת הווידאו שישמשו להרצה. אולם, בשל האוריינטציה הגרפית של התוכנה, מוטלת עליך, כמתכנת, אחריות נוספת כשאתה יוצר יישומים לחלונות. כפי שתיווכח בהמשך, מוקדשים פרקים רבים בספר זה לניהול נכון של המסך.

נתבונן עתה בכמה מהתכונות החשובות ביותר של חלונות 9x.

2.2 מבט אל שולחן העבודה

למעט חריגות ספורות, מטרת ממשק משתמש המבוסס על חלונות היא ליצור תמונת **שולחן עבודה** ממשי על פני המסך. על שולחן עבודה טיפוסי תמצא בוודאי ערימת ניירות, בדרך כלל תראה גם את קצוות הדפים שמלמטה. בסביבה החלונאית, הסביבה המקבילה לשולחן העבודה היא ה**מסך**. הדפים והניירות שעל השולחן מיוצגים על ידי החלונות השונים שעל המסך. כשאתה עובד על שולחן אמיתי, אתה מזיז את הניירות, ומדי פעם משנה את הסדר שלהם בערימה, כך שכל פעם נמצא דף אחר בראש הערימה. תוכנת חלונות מאפשרת לך לבצע פעולות דומות על המסך, בעזרת החלונות השונים. על ידי בחירה בחלון, אתה הופך אותו לחלון הפעיל, כלומר, מניח אותו בראש הערימה של כל החלונות הפתוחים באותה עת. בקצרה, חלונות מאפשרת לך לשלוט במסך ולהזיז עליו פריטים באותה דרך שבה אתה מזיז דברים על שולחן העבודה שלך.

דגם שולחן העבודה הוא הבסיס של ממשק המשתמש בחלונות, אך אין זו מתכונת המחייבת בהכרח את התוכניות עצמן. למעשה, אחד החידושים בחלונות הוא תוספת מרכיבי ממשק שונים המחקים סוגים אחרים של התקנים נפוצים, כגון בקרת גלישה, בקרת סיבוב, רשימות עץ וסרגלי כלים. כפי שתיווכח, מספקת חלונות למתכנת מערך מגוון של תכונות, דבר שמאפשר לכל אחד לבחור את המאפיינים והכלים המתאימים ביותר ליישום שבו הוא עוסק.

העכבר

תוכנת חלונות 9x מאפשרת שימוש בעכבר כמעט לכל פעולות הבקרה, הבחירה והציור. לומר שהתוכנה **מאפשרת** שימוש בעכבר היא המעטה מופלגת, כי הממשק של חלונות **תוכנן במיוחד לעכבר**. למעשה, הוא **מאפשר**, ברוב טובו, את השימוש במקלדת! בהחלט אפשר ליצור יישום המתעלם מהעכבר, אולם, זו תהיה הפרה של אחד מעקרונות העיצוב הבסיסיים של חלונות.

סמלים ומפות-סיביות

חלונות מעודדת ומפשטת את השימוש בסמלים (Icons) ובמפות-סיביות (Bitmaps - תמונות גרפיות). תפיסה זו מבוססת על האימרה "תמונה אחת שווה אלף מילים".

הסמל הוא תמונה זעירה המייצגת פעולה או תוכנית מסוימת. ככלל, ניתן לבחור בפעולה, או בתוכנית, על ידי בחירה בסמל. מפת-סיביות משמשת פעמים רבות להעברת מידע למשתמש בצורה מהירה ופשוטה, אך ניתן להשתמש במפות אלו גם כפריטים בתפריט.

תפריטים, סרגלי כלים, סרגלי סטטוס ותיבות דו-שיח

פרט לחלונות הרגילים, מכילה חלונות 9x גם כמה סוגים של חלונות מיוחדים, הנפוצים ביותר הם:

- ☆ **תפריט** (Menu) הוא חלון מיוחד, המכיל רשימה שממנה בוחר המשתמש פריט מסוים. אולם, במקום שתצטרך לכלול בתוכנית שלך את הפונקציות המתאימות לפריטי התפריט השונים, תוכל ליצור תפריט תקני בעזרת פונקציות מובנות שיתאימו לפריטי התפריט השונים.
- ☆ **סרגל כלים** (Tool bar) הוא ביסודו תפריט מיוחד, המציג את האפשרויות שלו באמצעות תמונות גרפיות קטנות (סמלים - Icons). המשתמש בוחר עצם מסוים על ידי לחיצה בעכבר על התמונה הרצויה לו.
- ☆ **סרגל הסטטוס** (Status Bar) הוא פס המופיע בתחתית חלון, המכיל מידע הנוגע למצב היישום.
- ☆ **תיבת דו-שיח** (Dialog Box) היא חלון מיוחד המאפשר תקשורת מורכבת יותר בין המשתמש והיישום מאשר באמצעות התפריט, או סרגל כלים. לדוגמה, היישום שלך יציג תיבת דו-שיח כדי לבקש שם קובץ. כמעט כל קלט שאינו נעשה דרך תפריט, נעשה באמצעות תיבת דו-שיח.

2.3 קשרי גומלין בין חלונות 9x לתוכנית שלך

כאשר אתה כותב תוכנית המיועדת למערכות הפעלה רבות, התוכנית היא זו שיוזמת את קשרי הגומלין עם מערכת ההפעלה. בתוכנית של DOS לדוגמה, התוכנית היא זו שמבקשת פריטים, כגון קלט ופלט. במילים אחרות, תוכנית שנכתבת במתכונת ה"מסורתית" קוראת למערכת ההפעלה. בחלונות 9x, הדברים הפוכים. דווקא מערכת ההפעלה היא זו הקוראת לתוכנית שלך. התהליך מתנהל כך: התוכנית שלך מחכה עד שהיא מקבלת **הודעה** (Message) מחלונות. לאחר שהתקבלה ההודעה, אמורה התוכנית שלך לנקוט את הפעולה המתאימה. התוכנית עשויה להפעיל פונקציה אחת, או יותר, מתוך API של חלונות כאשר היא מגיבה להודעה, אך מערכת ההפעלה (חלונות 9x)

יוזמת את הפעילות. יותר מכל, קשרי הגומלין עם חלונות, המבוססים על הודעות, הם המכתיבים את המתכונת והצורה הכללית של כל התוכניות לחלונות 9x.

חלונות עשויה לשלוח לתוכנית שלך סוגים רבים של הודעות. לדוגמה, לחיצת עכבר על אחד החלונות של התוכנית שלך תגרום לחלונות 9x לשלוח לתוכנית הודעה שמקורה בלחיצת עכבר. הודעה מסוג אחר תישלח כל פעם שיש ליצור מחדש את אחד החלונות בתוכנית. הודעה מיוחדת אחרת נשלחת כל פעם שהמשתמש מקיש על מקש כלשהו, כשהתוכנית נמצאת במוקד הקלט. כדאי לזכור היטב עובדה אחת: ככל שהדבר נוגע לתוכנית שלך, ההודעות מגיעות בצורה אקראית. מסיבה זו תוכניות של חלונות דומות לתוכניות המופעלות בשיטת הפסיקות. אין דרך לדעת מה תהיה ההודעה הבאה.

2.4 ממשק תכנות יישומים - API של Win32

הגישה לסביבת חלונות נעשית באמצעות ממשק מבוסס על קריאות, המכונה **ממשק תכנות יישומים** (API). API מכיל כמה מאות פונקציות שהתוכנית שלך מפעילה אותן לפי הצורך. הפונקציות שמכיל API מבצעות את כל שירותי המערכת הכלולים בחלונות. ל-API מערך-משנה המכונה **ממשק התקנים גרפיים** - **GDI** (Graphics Device Interface), שהוא אותו חלק מתוכנת חלונות המעניק תמיכה לגרפיקה **שאינה תלוית-התקן**. פונקציות GDI הן אלו המאפשרות להריץ יישומים של חלונות על מיגוון רחב של פריטי חומרה.

התוכניות במערכת חלונות מבוססות על API של Win32. ממשק Win32 תומך במיעון כתובות בשיטת 32 סיביות בעוד ש-Win16 תמך רק בשיטת הזיכרון המקוטע בן 16 סיביות.

בגלל ההבדל בשיטת המיעון, חלק מהפונקציות הישנות יותר ב-API הורחבו, כדי שתוכלנה לקבל פרמטרים של 32 סיביות ולהחזיר ערכים של 32 סיביות. כמו כן, היה צורך לשנות כמה מפונקציות API כדי להתאימן למבנה לפי שיטת 32 סיביות. פונקציות חדשות התוספו ל-API כדי לתמוך בגישה החדשה לריבוי משימות, במרכיבי הממשק החדשים וביתר התכונות המשופרות שהגיעו בחלונות 9x.

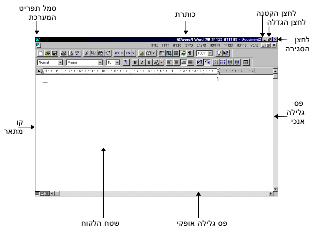
כיון שחלונות תומכת במיעון כתובות מלא בשיטת 32 סיביות, ההיגיון מכתיב מספרים שלמים שיהיו אף הם באורך של 32 סיביות. פירוש הדבר, שהסוגים Int ו-Unasigned הם באורך 32 סיביות. אם אתה רוצה להשתמש במספר שלם באורך 16 סיביות, חובה להגדיר אותו כ-Short (חלונות 9x מספקת לך שמות Typedef ניתנים להמרה עבור סוגים אלה, כפי שתיווכח מייד).

השלכה נוספת של מיעון הכתובות בשיטת 32 סיביות היא, שאין עוד צורך להגדיר מצביעים כ - Near או Far. לכל מצביע יש גישה לכל חלק מהזיכרון. בחלונות 9x, אין שום הגדרה ל-Near או ל-Far. כלומר, אתה יכול להשאיר את Near ו-Far בתוכניות שאתה מייבא אל תוך חלונות, אך לא תהיה להם השפעה.

2.5 מרכיבי חלון בתוכנית Windows

אולי הצלחת לנחש: אבן היסוד של בניית תוכנית לסביבת Windows זהו חלון אחד או יותר, או תיבות דו-שיח (שגם הן סוג מיוחד של חלון). למעשה, כמעט כל עצם שנמצא בחלון הוא גם כן חלון מסוג מסוים. בסעיפים הבאים תבין טוב יותר שהתוכניות משתמשות בחלונות דומים למדי, למרות שהם נגזרים ממקומות שונים. במה שקשור לסעיף מסוים זה, חשוב שתדע את המרכיבים של **חלון רגיל** (Standard Window) - במילים אחרות, מה שהמשתמש בתוכנית מזהה כ**חלון**.

ככלל, Windows בונה כל "חלון רגיל" משבעה חלקים בסיסיים. באפשרותך לפרק כל אחד מחלקים אלה לחלקים יותר קטנים, דבר שתעשה ברוב הסעיפים הבאים. אבל, חשוב שתבין את צורת החלון באופן כללי לפני שתתחיל בניתוח חלקים קטנים אלה, שכולם יחד יוצרים את כל אחד ממרכיבי החלון. בתרשים 2.1 מובאת תמונה של חלון רגיל, על כל מרכיביו.



תרשים 2.1: המרכיבים של חלון רגיל

חלון המסגרת (Frame Window) הוא **המכולה** (Container) של כל מה שנמצא בחלון. כמו שנלמד בהמשך, אפשר ליצור חלונות מסגרת רבים ומסוגים שונים. חלון המסגרת השימושי ביותר הוא זה שמאפשר לשנות את הגודל שלו, בדומה לזה שמוצג בתרשים 2.1. בתוכניות שלך תטפל במסגרת ותקבל הודעות רבות מהמסגרת, כמו ההודעה לשינוי גודל החלון. הסעיפים הבאים דנים בחלון המסגרת באופן מפורט יותר.

שורת הכותרת (Title Bar) מציגה למשתמש מידע על התוכנית. שורת הכותרת נמצאת לרוחב החלון בקצה הסמוך לגבולו העליון. שורת הכותרת מאפשרת זיהוי תוכן החלון ומאפשרת למשתמש לבצע פעולות חלון רבות. שורת הכותרת משמשת כנקודת האחיזה בחלון כשרוצים להזיז אותו ונמצא בה גם **תפריט המערכת** (System Menu), והלחצנים האלה: **הקטנה** (Minimize), **הגדלה** (Maximize), **שחזור** (Restore) ו**סגירת חלון** (Close Window). בסעיפים הבאים נלמד שרכיבי שורת הכותרת משתנים כתלות ביישום שיוצר את החלון שבו נמצאת שורת הכותרת.

הלחצנים השונים להקטנה, להגדלה ולסגירת החלון, שנמצאים בשורת הכותרת, חשובים מאוד, ולכן עליך לקחת אותם בחשבון כמרכיבי החלון. לחצנים אלה מאפשרים לך שליטה בגודל החלון וסגירתו כשלא צריך אותו יותר.

שטח הלקוח (Client Area) של החלון הוא חלק החלון אשר באפשרותך לתכנן את תוכנו כרצונך ולהתאים אותו לדרישותיך המיוחדות, כמו למשל קלט נתונים מהמשתמש. במילים אחרות, שטח הלקוח הוא אותו חלק של החלון אשר בו מתרחשת מרבית הפעילות של התוכנית. לדוגמה, אם אתה עובד עם מעבד תמלילים כמו Word של מיקרוסופט, שטח הלקוח הוא אזור החלון שבו אתה כותב ועורך את המסמך.

פסי הגלילה (Scroll Bars) מאפשרים למשתמש לנוע בחלון שמאלה וימינה ולמעלה ולמטה. לדוגמה, תוכל להשתמש בתוכניות שלך בפסי הגלילה כדי לאפשר למשתמש לגלול את הקלט על פני החלון. שימוש נוסף בפסי הגלילה בתוכניות הוא לאפשר למשתמש לנוע בתוך מסמך ששמר לפני כן בדיסק, כמו שהוא משתמש בפסי הגלילה בתוכנה כמו Word. פסי הגלילה הם כלי מאוד חשוב לניווט בתוכניות Windows.

שורת התפריט (Menu Bar) היא מרכיב שכיח מאוד בחלונות אב (Parent Windows), אך בדרך כלל לא נמצא ברוב חלונות הבנים. תוכניות Windows משתמשות בשורת התפריט כדי לספק למשתמש אפשרויות פקודה רבות המתאימות לתוכנית. כמו שנלמד בהמשך, רוב תוכניות Windows מכילות לפחות את התפריטים **קובץ** (File) ו**עזרה** (Help). תוכניות Windows מורכבות יותר מכילות גם 10 תפריטים ויותר, ויש תפריטים שיש בהם אפילו 20 אפשרויות בחירה של פקודות. ככל שהתוכניות שלך יהפכו להיות יותר מסובכות ומורכבות, כך גם התפריטים שלך יהיו מורכבים יותר. תרשים 2.2 מציג את שורת התפריט של Microsoft Word.



תרשים 2.2: שורת התפריט של Microsoft Word.

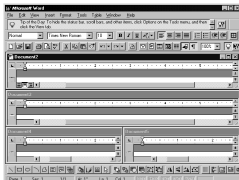
שורת המצב (Status Bar) נמצאת בתחתית רוב חלונות האב, אך בדרך כלל לא נמצא אותה ברוב חלונות הבנים. תוכניות Windows משתמשות בשורת המצב כדי לספק למשתמש מידע שמפרט את מצב המשתמש בתוכנית - מקומו במסמך, בשורה, וכדומה. תרשים 2.3 מציג את שורת המצב של Word, שמספקת מידע חשוב על מיקומו הנוכחי של המשתמש במסמך.



תרשים 2.3: שורת המצב של Microsoft Word.

2.6 חלון-אב וחלון-בן (Parent and Child Windows)

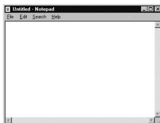
בקטע הקודם למדת, שרוב חלונות האב מכילים את שורת התפריט ושורת המצב, אך רוב חלונות הבן אינם מכילים מרכיבים אלה. אולי אינך יודע עדיין מהו **חלון-אב** ומהו **חלון-בן**, אך נקדיש לכך הסבר קצר. מרבית תוכניות Windows תומכות **בממשק מרובה מסמכים** (Multiple Document Interface) ובקיצור **MDI**. ממשק זה מאפשר לתוכנית אחת להחזיק ולהציג רכיבים רבים בחלון אחד. לדוגמה, תרשים 2.4 מציג את תוכנית Word שבחלון שלה מוצגים ארבעה מסמכים פתוחים.



תרשים 2.4: חלון-האב של Word עם ארבעה חלונות-בן פתוחים.

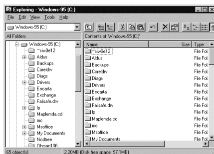
דבר זה דומה יותר להיררכיית המחלקות שלמדת והשתמשת בה בתוכניות ++C. כל חלון נגזר מחלון בסיסי מסוים ולכן, לכל חלון יש חלון-אב. בתרשים 2.4, חלונות הבן הם החלונות הפנימיים שבמסמך, וחלון האב הוא החלון של Word. כמו כן, אתה יכול לחשוב על החלון של Word כחלון-בן ואת **חלון שולחן העבודה** (Desktop Window) של Windows תוכל לראות כחלון האב שלו. לחלון שולחן העבודה אין חלון-אב.

בתרשים הקודם, חלון האב תומך בממשק מרובה מסמכים, שמאפשר לחלון האב שיהיה לו חלונות בן רבים. תוכניות Windows אחרות תומכות **בממשק מסמך בודד** (Single Document Interface) ובקיצור SDI. ממשק זה מאפשר לחלון האב להחזיק רק בחלון אחד, כמוצג בתרשים 2.5.



תרשים 2.5: תוכנית פנקס הרשימות (Notepad) של Windows היא בעלת ממשק מסמך בודד.

לבסוף, רבות מתוכניות Windows תומכות בגרסאות מיוחדות ושונות של ממשק מסמך בודד, שידועות בשם **ממשק בסגנון סייר Windows** (Explorer-Style Document Interface). המתכנתים קוראים לגרסאות המיוחדות האלו בשם זה, מכיון שהם מעצבים את הממשק על פי הממשק של סייר Windows. ממשק זה דומה לממשק מסמך בודד בכל התכונות, מלבד העובדה ש**חלון המסמך** (Document Window) היחיד מתפצל באמצע, והדבר מאפשר לתוכנית להציג קבוצות נתונים מיוחדים בתצוגה יחידה (Single View) ובדרך קלה יותר. תרשים 2.6 מציג את **סייר Windows** (Windows Explorer) ואת סגנון הממשק המיוחד שלו.



תרשים 2.6: סייר Windows וממשק בסגנון הסייר.

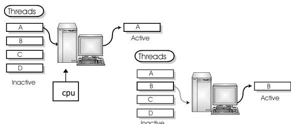
בתוכניות שלך תנהל בדרך כלל חלונות-בן רבים בתוך חלון-אב יחיד. ככל שתתקדם בתכנון חלונות, תכיר יותר את ההבדלים החשובים שבין הגרסאות השונות של הממשקים.

2.7 יישומי חלונות 9x, עקרונות בסיסיים

בטרם ניגש לפיתוח השלד של יישומי חלונות, ראוי לדון בכמה מהמושגים הבסיסיים המשותפים לכל התוכניות הפועלות במערכת חלונות 9x.

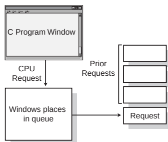
הקדמה למטלות (Threads)

אנו מניחים שיש לך ניסיון בתכנות C++ בסביבת DOS. על כן, אתה בוודאי יודע ש-DOS תומכת בהרצת תוכנית אחת בלבד בכל זמן נתון, ורק תוכנית זו נמצאת בזיכרון באותו זמן, אך לא כך הדבר בסביבת העבודה Windows שאינה כפופה לאילוץ זה. למעשה, Windows מגבילה את מספר התוכניות שמחשב מסוגל להפעיל בו-זמנית רק אם אין מספיק זיכרון לפתוח תוכניות נוספות. היא מנהלת את התוכניות האלו בזיכרון בשיטת ה**מטלות**. **מטלה** (Thread) היא, למשל, בקשה של התוכנית להשתמש ב**מעבד** (CPU) של המחשב. כשיש לך מספר תוכניות שפועלות בו-זמנית, מערכת ההפעלה תנהל מטלה אחת או יותר עבור כל אחת מתוכניות אלו. Windows מציבה כל מטלה בתור (רשימה) של מטלות הממתינות לביצוע. אחר כך, בהסתמך על העדיפות שיש למטלה, Windows 'שולפת' מטלה מהתור ומקצה לה זמן מעבד לביצוע ההוראות של המטלה. עדיפות המטלה קובעת היכן Windows מציבה אותה ביחס למטלות אחרות שבתור. תרשים 2.7 מציג תיאור גרפי פשוט לדרך שבה Windows מנהלת מטלות.



תרשים 2.7: ה**מעבד** מטפל בסדרת מטלות.

נחזור ונדון במטלות בהרחבה בסעיפים שנלמד מאוחר יותר. אבל, מה שעליך לדעת כרגע הוא שמטלה היא הישות ש-Windows מקצה עבורה זמן מעבד להרצת הוראות התוכנית הכלולות בה. Windows מארגנת את המטלות בהסתמך על העדיפות שלהן והמעבד מבצע כל מטלה לפי התור. בתרשים 2.8 מוצג מודל פשוט שמראה איך התוכנית מבקשת מהמעבד לבצע פעולה כלשהי, ומקבלת ממנו את התוצאה בסיום העיבוד.

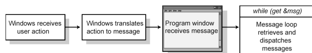


תרשים 2.8: סכמה פשוטה לעיבוד מטלות על ידי Windows.

Windows מחזירה את התוצאות לתוכניות שלך בסיום העיבוד בפורמט הקרוי **הודעה** (message).

2.8 הודעות התוכנית (Messages)

האמצעי העיקרי שמשמש לתקשורת בין Windows והתוכניות הכתובות לסביבת Windows הן **הודעות** (Messages). הדבר פשוט למדי. בכל פעם שמתבצעת פעולה מסוימת, Windows מגיבה על ידי משלוח הודעה אל עצמה, או אל תוכנית אחרת. לדוגמה, כשהמשתמש לוחץ על העכבר בחלון התוכניות שלך, Windows 'קולטת' את לחיצת העכבר ושולחת הודעה לתוכנית שלך, בה היא מודיעה לה על כך שהמשתמש לחץ על העכבר בחלון התוכניות במקום מסוים. כשהתוכנית שלך מקבלת את ההודעה, היא מתחילה תהליך עיבוד שמתאים להודעה שנקלטה. אם ההודעה אינה חשובה לתוכנית, היא צריכה להתעלם ממנה. להבנת מודל ההודעה של Windows יותר טוב, התבונן בתרשים 2.9, שמראה את המודל באופן מופשט, בצורה ליניארית.



תרשים 2.9: סכמה פשוטה למודל ההודעות של Windows.

כשכותבים תוכניות בסביבת Windows השגרות החשובות ביותר בתוכניות תהיינה אלו שמקבלות ומעבדות הודעות ממערכת ההפעלה. לדוגמה, קטע הקוד הבא של הפונקציה WndProc מציג עיבוד פשוט של הודעה ופונקציה שמגיבה להודעה שלהלן.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
LPARAM lParam)
 {
   switch(uMsg)
    {
      case WM_COMMAND:
         switch(LOWORD(wParam))
          {
            case IDM_TEST :
               break;
            case IDM_EXIT :
               DestroyWindow(hWnd);
               break;
          }
         break;
      case WM_DESTROY :
         PostQuitMessage(0);
         break;
      default:
         return (DefWindowProc(hWnd, uMsg, wParam, lParam));
    }
   return(0L);
 }
```

באופן כללי, קטע הקוד משתמש במשפט **switch** כדי לקבוע את סוג ההודעה שהתקבלה. ככל שתתקדם בלימוד הסעיפים, תבין בקלות את העיבוד שנעשה בפונקציה WndProc. בקטע הקוד הזה, הפונקציה בודקת את ההודעה WM_COMMAND או את ההודעה WM_DESTROY, או שהיא מוסרת את ההודעה לפונקציית ברירת המחדל שמטפלת בהודעות. הפונקציה פועלת על פי סוג ההודעה שהיא מקבלת.

2.9 WinMain()

כל התוכניות המיועדות למערכת חלונות 9x מתחילות את ההרצה בקריאה לפונקציה ()WinMain (גרסאות קודמות של חלונות אינן מכילות פונקציה זו). לפונקציה ()WinMain מספר תכונות מיוחדות, המבדילות אותה משאר הפונקציות ביישומים שלך. יש להדר אותה בנוהל הקריאה של WINAPI (בוודאי תיתקל גם במונח APIENTRY, אשר זהה למונח WINAPI). לפי ברירת המחדל, הפונקציות בתוכניות שלך הכתובות ב-C או ב- ++C משתמשות בנוהל הקריאה של C. ניתן להדר פונקציה, כך שתוכלו להשתמש בנוהל קריאה אחר; אחת החלופות הנפוצות היא פסקל. מסיבות טכניות שונות, נוהל הקריאה המשמש את חלונות 9x להפעלת ()WinMain הוא הנוהל של WINAPI. הערך החוזר של פונקציה זו אמור להיות מסוג int.

2.10 פונקציית החלון

כל תוכנית של חלונות 9x חייבת להכיל פונקציה מיוחדת **שאינה** מופעלת על ידי התוכנית שלך, אלא על ידי חלונות 9x. פונקציה זו מכונה בדרך כלל **פונקציית החלון** (Window Function) או **נוהל החלון** (Window Process). חלונות קוראת לפונקציה זו כאשר היא רוצה להעביר הודעה לתוכנית שלך, כלומר, זהו הערוץ שבו משתמשת חלונות 9x כדי לתקשר עם התוכנית. פונקציית החלון מקבלת את ההודעה בפרמטרים שלה. חובה להגדיר את כל פונקציות החלונות כפונקציות המחזירות ערכים מסוג LRESULT CALLBACK. הסוג LRESULT הוא מחרוזת typedef, שבזמן כתיבתה היא שם חלופי למספר שלם ארוך. נוהל הקריאה CALLBACK משמש באותן פונקציות שיופעלו בידי חלונות 9x. במונחים חלונאיים, כל פונקציה המופעלת בידי חלונות, מכונה **פונקציית משוב** (Callback).

נוסף לקבלת הודעות הנשלחות על ידי חלונות 9x, מתפקידה של פונקציית החלון ליזום כל פעולה שמכתיבות ההודעות הללו. בדרך כלל, מכיל גוף הפונקציה משפט switch המקשר תגובה מוגדרת לכל הודעה שהתוכנית אמורה להגיב עליה. התוכנית שלך אינה מגיבה לכל ההודעות שחלונות 9x שולחת. כשמדובר בהודעות שאינן מעניינות של התוכנית שלך, הנח לחלונות 9x לבצע את פעולת ברירת המחדל. כיון שחלונות 9x מסוגלת להפיק מאות הודעות, רובן עוברות עיבוד על ידי חלונות 9x עצמה, ולא על ידי התוכנית שלך. יתר על כן, כל הודעה מקושרת עם כל המידע הנוסף שהיא עשויה להזדקק לו.

2.11 סגנונות של חלונות

בעת שהתוכנית שלך מתחילה לפעול תחת חלונות 9x, חובה עליה להגדיר **מחלקת חלונות** (Window Class). המונח **מחלקה** אינו משמש במשמעות המקובלת ב- ++C, אלא פירושו קרוב יותר ל**סגנון** (Style), או **סוג** (Type). כשאתה מגדיר את סגנון החלון, אתה מודיע לחלונות 9x על צורת החלון ועל תפקודו. הגדרת סגנון החלון אינה הפעולה היוצרת את החלון. על מנת ליצור את החלון בפועל, עליך לבצע כמה צעדים נוספים.

2.12 לולאת ההודעות

כפי שהסברנו קודם, חלונות 9x מתקשרת עם התוכנית שלך באמצעות הודעות. כל יישום של חלונות 9x חייב ליצור **לולאת הודעות** (Message Loop) בתוך הפונקציה ()WinMain. לולאה זו קוראת כל הודעה נכנסת מתור ההודעות של היישום, ושולחת את ההודעה בחזרה לחלונות 9x. חלונות 9x קוראת לפונקציית החלון של התוכנית שלך, כשההודעה הזו משמשת לה כפרמטר. ייתכן שהדבר נראה לך כשיטה מסורבלת להעברת הודעות, אך זו הדרך שבה חייבות לפעול כל התוכניות של חלונות. הסיבה לנוהל זה היא הצורך להחזיר את השליטה לחלונות 9x, כך שה-scheduler יוכל להקצות זמן מעבד, לפי הנחוץ, ולא ימתין שהיישום שלך יסיים את פלח הזמן שלו.

2.13 סוגים של נתוני חלונות

כפי שתראה מייד, תוכניות שפועלות בחלונות 9x אינן מרבות להשתמש ב**סוגי הנתונים** הרגילים של C/C++, כגון int או char *. במקום זה, נעשית לכל סוגי הנתונים המשמשים את חלונות 9x הסבה ל-typedef. ההסבה נעשית בתוך הקובץ windows.h ו/או הקבצים הקשורים אליו. הקובץ windows.h מסופק על ידי מיקרוסופט, או בורלנד (וכל חברה אחרת המייצרת מהדרי C/C++ לחלונות), וחובה לכלול אותו בכל תוכניות חלונות 9x. בין הסוגים הנפוצים ביותר נמנים HANDLE, HWND, BYTE, WORD, DWORD, UINT, LONG, BOOL, LPSTR ו-LPCSTR. הסוג HANDLE הוא מספר שלם בן 32 סיביות, המשמש כידית. קיימים מספר סוגים של ידיות, אבל כולם בגודל זהה ל-HANDLE. **ידית** (Handle) היא ערך המגדיר משאב כלשהו.

כל סוגי הידיות מתחילים באות H. לדוגמה, הערך HWND הוא מספר שלם בן 32 סיביות, המשמש ידית לחלון. BYTE הוא תו לא מסומן בן 8 סיביות. WORD הוא מספר שלם קצר ולא מסומן בן 16 סיביות. DWORD הוא מספר שלם ארוך ולא מסומן. UINT הוא מספר שלם לא מסומן בן 32 סיביות. LONG הוא שם אחר ל-long. נתונים מסוג BOOL הם מספרים שלמים, המשמשים לציון ערכי אמת או שקר (True/False). LPSTR הוא מצביע אל מחרוזת, ו-LPCSTR הוא מצביע מסוג const אל מחרוזת.

נוסף לסוגים הבסיסיים שתוארו, מגדירה חלונות 9x כמה מבנים. השניים הנחוצים לנו לבניית שלד התוכנית הם MSG ו-WNDCLASS. מבנה MSG מכיל הודעה של חלונות 9x, ואילו WNDCLASS הוא מבנה המגדיר את סגנון החלון. מבנים אלה יידונו בהמשך הפרק.

2.14 תוכנית מסגרת לחלונות 9x

לאחר שעסקנו ברקע הנחוץ, הגיע הזמן לפתח יישום פשוט לחלונות 9x. כפי שכבר נאמר, כל תוכניות חלונות 9x מכילות מרכיבים משותפים מסוימים. בסעיף זה נפתח תוכנית מסגרת לחלונות 9x, שתכיל את כל המאפיינים הנחוצים האלה. בממלכת התכנות החלונאי נפוץ מאוד השימוש בשלדים, או במסגרות של יישומים, כיון שיצירת תוכנית של חלונות כרוכה ב״מחיר כניסה״ גבוה למדי. שלא כמו ב-DOS, שבה התוכנית הפשוטה ביותר מכילה כ-5 שורות, התוכנית המינימלית של חלונות מכילה 50 שורות בקירוב.

תוכנית פשוטה של חלונות 9x מכילה שתי פונקציות: ()WinMain ופונקציית חלון. מתפקידה של הפונקציה ()WinMain לבצע את השלבים הכלליים הבאים:

1. להגדיר את סגנון החלון.
2. לרשום את סגנון החלון בתוך חלונות 9x.
3. ליצור חלון בסגנון שהוגדר.
4. להציג את החלון.
5. להתחיל בהרצת לולאת ההודעות.

תפקידה של פונקציית החלון הוא להגיב לכל ההודעות הרלוונטיות. כיון שתוכנית השלד אינה עושה דבר פרט להצגת החלון שלה, ההודעה היחידה שהיא נדרשת להגיב עליה היא ההודעה המורה ליישום שהמשתמש - סיים את התוכנית.

לפני שנדון בפרטים, עיין בתוכנית **generic** המובאת להלן (התוכנית נמצאת בתקליטור בתיקיה books\59285\chap02), שהיא שלד פשוט ומזערי לתוכנית חלונות 9x. תוכנית זאת יוצרת חלון רגיל הכולל כותרת. החלון מכיל גם את תפריט המערכת ועל כן ניתן להקטינו, להגדילו, להזיזו, לשנות את גודלו ולסוגרו. החלון מכיל גם את לחצני ההקטנה, ההגדלה והסגירה התקניים.

```
#include <windows.h>
#include "generic.h"
#if defined (win32)
   #define IS_WIN32 TRUE
#else
   #define IS_WIN32 FALSE
#endif

HINSTANCE hInst;         // current instance
LPCTSTR lpszAppName = "Generic";
LPCTSTR lpszTitle =   "Generic Application";
BOOL RegisterWin95(CONST WNDCLASS* lpwc);

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
 {
   MSG msg;
   HWND hWnd;
   WNDCLASS wc;

   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = (WNDPROC)WndProc;
   wc.cbClsExtra     = 0;
   wc.cbWndExtra     = 0;
   wc.hInstance      = 0;
   wc.hIcon          = LoadIcon(hInstance, lpszAppName);
   wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName   = lpszAppName;
   wc.lpszClassName  = lpszAppName;
```

```
   if(!RegisterWin95(&wc))
      return false;
   hInst = hInstance;
   hWnd = CreateWindow (lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                       );
   if(!hWnd)
      return false;
   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
   while(GetMessage(&msg, NULL, 0,0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
   return(msg.wParam);
 }

BOOL RegisterWin95(CONST WNDCLASS* lpwc)
 {
   WNDCLASSEX wcex;

   wcex.style           = lpwc->style;
   wcex.lpfnWndProc     = lpwc->lpfnWndProc;
   wcex.cbClsExtra      = lpwc->cbClsExtra;
   wcex.cbWndExtra      = lpwc->cbWndExtra;
   wcex.hInstance       = lpwc->hInstance;
   wcex.hIcon           = lpwc->hIcon;
   wcex.hCursor         = lpwc->hCursor;
   wcex.hbrBackground   = lpwc->hbrBackground;
   wcex.lpszMenuName    = lpwc->lpszMenuName;
   wcex.lpszClassName   = lpwc->lpszClassName;
   wcex.cbSize          = sizeof(WNDCLASSEX);
```

```
   wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");
   return RegisterClassEx(&wcex);
 }

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam)
 {
   switch(uMsg)
    {
      case WM_COMMAND:
         switch(LOWORD(wParam))
          {
            case IDM_TEST :
               break;
            case IDM_EXIT :
               DestroyWindow(hWnd);
               break;
          }
         break;
      case WM_DESTROY :
         PostQuitMessage(0);
         break;
      default:
         return (DefWindowProc(hWnd, uMsg, wParam, lParam));
    }
   return(0L);
 }
```

נסקור עתה את התוכנית צעד אחר צעד.

ראשית, כל תוכניות חלונות 9x חייבות לכלול את קובץ הכותר windows.h. כאמור, קובץ זה (יחד עם קובצי העזר שלו), מכיל את אבות הטיפוס של פונקציות API, וגם פקודות מאקרו, הגדרות וסוגים שונים, המשמשים את חלונות. לדוגמה, ההגדרות של סוגי הנתונים HWND ו-WNDCLASS כלולות בקובץ windows.h.

ההכרזות מגדירות אחר כך את lpszAppName ואת lpszTitle, שנראה במבט ראשון כמערך תווים, או אולי כמו משתני מחרוזות. תשתמש בסוג LPCSTR (סוג שמוגדר תחת Windows) כדי להחזיק מצביעים למחרוזות לקריאה בלבד.

כשהמהדר מהדר את התוכנית, התוכנית ממירה למעשה את כל הכרזות LPCSTR להכרזות מסוג: const char FAR*. ובכן, כמו שאתה רואה, השימוש ב-LPCSTR יותר קל להבנה ולהדפסה מאשר התחליף שלו.

לבסוף, התוכנית מכריזה על אבטיפוס לפונקציה RegisterWin95. הפונקציה RegisterWin95 מקבלת פרמטר מסוג WNDCLASS ומחזירה ערך בוליאני שמצביע על הצלחה או כישלון.

פונקציית החלון שהתוכנית משתמשת בה נקראת ()WndProc, המוגדרת כפונקציית משוב, כי זוהי הפונקציה שחלונות מפעילה כדי לתקשר עם התוכנית.

הפונקציה WinMain

בתוכנית **generic** מן הסעיף הקודם, הפונקציה הראשונה שבתוכנית היתה WinMain. כמו שלמדת, הפונקציה WinMain הינה הפונקציה המקבילה של Windows לפונקציה main שכל תוכניות C ו- ++C שלך השתמשו בה כפונקציה ראשית לעיבוד בסביבת העבודה DOS. הפונקציה WinMain, שונה במספר דברים חשובים, אשר ההבדל בהכרזת הפונקציה הוא אחד מהם, כפי שמוצג להלן :

```
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine, int nCmdShow)
```

כפי שתוכל לראות, הפונקציה WinMain מחזירה ערך int, כמו שעושות תוכניות ++C רבות אחרות. ובכן, זהו המקום שבו הן דומות (ואמנם, כמו שתלמד אחר כך, משפט הכותרת של הפונקציה WinMain מבצע עיבוד דומה לזה של הפונקציה main). מילת המפתח APIENTRY מראה שהמשתמש יכול להפעיל את התוכנית מתוך Windows בלבד. טבלה 2.1 מפרטת את הפרמטרים שחובה על התוכניות שלך להעביר לפונקציה WinMain.

טבלה 2.1: הפרמטרים שצריך להעביר לפונקציה WinMain.

שם הפרמטר	סוג הפרמטר	תיאור
hinstance	HINSTANCE	ידית המופע של היישום. לכל מופע של יישום יש ידית מיוחדת. התוכניות שלך ישתמשו בערכי hinstance כארגומנט למספר פונקציות של Windows. גם אפשר להשתמש בערך hinstance כדי להבדיל בין מופעים רבים של יישום מסוים כלשהו.
HPrevInstance	HINSTANCE	ידית המופע הקודם של היישום. עבור המופע הראשון ערך זה הוא NULL. עבור Windows 9x ערך זה תמיד יהיה NULL.

שם הפרמטר	סוג הפרמטר	תיאור
LpCmdLine	LPSTR	זהו מצביע מסוג far לשורת הפקודה שמסתיימת ב-NULL. הגדר את הערך lpCmdLine כשאתה מפעיל את היישום מתוך מנהל היישומים (Program Manager) או על ידי קריאה ל-WinExec. שים לב, תחת Windows 9x פרמטר זה הוא מצביע לשורת הפקודה כולה, ולא מערך מצביעים לכל פרמטר (לכן, חובה על התוכניות שלך לפרש את שורת הפקודה לפני שמתחילה לטפל בה).
NCmdShow	int	מספר שלם שקובע כיצד יוצג חלון היישום על המסך. צריך להעביר ערך זה ל-ShowWindow.

הפעולה הראשונה שהפונקציה WinMain מבצעת בתוכנית **generic** היא להגדיר משתנה מסוג MSG, משתנה מסוג HWND, ומשתנה מסוג WNDCLASS, כפי שמוצג להלן:

```
MSG msg;
HWND hWnd;
WNDCLASS wc;
```

בתוך הפונקציה נוצרים שלושה משתנים. המשתנה hWnd יכיל את הידית לחלון התוכנית. משתנה-המבנה msg יכיל הודעות לחלונות, ומשתנה-המבנה wc ישמש להגדרת המחלקה של החלון. ההוראות שבאות אחר כך בתוכנית **generic** הן הוראות השמה לאיברי המשתנה wc.

הגדרת סגנון החלון

חובה עליך לרשום את סגנון החלון ב-Windows לפני שתוכל להשתמש בו כדי ליצור את החלונות. לצורך זה תשתמש בפונקציית API בשם RegisterClass. בתוכניות שלך תשתמש בפונקציה RegisterClass כמו שתוכל לראות באבטיפוס שלהלן:

```
ATOM RegisterClass(CONST WNDCLASS* lpwc);
```

תוכל לראות שהפונקציה RegisterClass מחזירה ערך מסוג ATOM. סוג זה הוא ערך של WORD שמתייחס למחרוזות תווים, שאין בה הבחנה בין אותיות רישיות לבין אותיות רגילות. העובדה שהערכים מסוג ATOM מתייחסים למחרוזות ללא הבחנה בין אותיות רישיות לרגילות, פירושו ש-"happy" זהה ל-"HAPPY" - שוויון אשר, כמו שאתה יודע, אינו נכון בדרך כלל ב- ++C. Windows שומרת את הערכים מסוג ATOM בטבלת ATOM וכך, הערך WORD שמכיל משתנה מסוג ATOM דומה במידה רבה לידית.

בנוסף להחזרת ערך מסוג ATOM, הפונקציה RegisterClass מקבלת פרמטר אחד - מצביע קבוע למבנה מסוג WNDCLASS. Windows מגדירה את המבנה WNDCLASS כפי

שמוצג להלן. התבונן במשפטי ההשמה הבאים שנלקחו מהתוכנית **generic**, שמאתחלים את מחלקת החלון הספציפית שהתוכנית משתמשת בה:

```
wc.style         = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc   = (WNDPROC)WndProc;
wc.cbClsExtra    = 0;
wc.cbWndExtra    = 0;
wc.hInstance     = hInst;
wc.hIcon         = LoadIcon(hInstance, lpszAppName);
wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wc.lpszMenuName  = lpszAppName;
wc.lpszClassName = lpszAppName;
```

משפט ההשמה הראשון אומר למערכת ההפעלה לצייר את כל החלון מחדש בכל פעם שהמשתמש משנה את גודלו האופקי או האנכי. משפט ההשמה השני אומר למערכת ההפעלה ***שפונקציית המשוב*** (Callback Function) זו הפונקציה WndProc. שני משפטי ההשמה הבאים מורים לפונקציה RegisterClass לא להקצות מקום נוסף, וההוראה wc.hInstance מורה למהדר להשתמש במופע הנוכחי של התוכנית.

שני משפטי ההשמה הבאים אחר כך, hIcon ו-hCursor, טוענים את הסמל והסמן של החלון.

בכל היישומים של חלונות חובה להגדיר צורת ברירת מחדל לסמן העכבר ולסמל היישום. אפשר להגדיר ליישום גירסה מותאמת במיוחד של משאבים אלה, או שניתן להשתמש באחד הסגנונות המובנים, כפי שעשינו בתוכנית המסגרת. סגנון הסמל נטען על ידי פונקציית API בשם ()LoadIcon, שהגדרתו מובאת להלן:

```
HICON LoadIcon(HINSTANCE hInst, LPCSTR lpszName);
```

פונקציה זו מחזירה ידית לסמל. כאן, הפרמטר hInst מציין את הידית למודול המכיל את הסמל, ואילו שם הסמל מצוין בפרמטר lpszName. כדי להשתמש באחד הסמלים הקיימים במערכת, עליך להשתמש בערך NULL עבור הפרמטר הראשון, ולציין את אחד משמות המאקרו הבאים עבור הפרמטר השני.

שם המאקרו לסמל	תיאור
IDI_APPLICATION	הסמל בהתאם לברירת המחדל
IDI_ASTERISK	סמל מידע/Information
IDI_EXCLAMATION	סמל בצורת סימן קריאה
IDI_HAND	סמל בצורת תמרור "עצור"
IDI_QUESTION	סמל בצורת סימן שאלה

✍ **הערה:** קיימים סמני מערכת נוספים. למידע נוסף פנה ל-MSDN.

כדי לטעון את סמן העכבר, הפעל את הפונקציה ()LoadCursor. להלן הגדרת הפונקציה:

```
HCURSOR LoadCursor(HINSTANCE hInst, LPCSTR lpszName);
```

הפונקציה מחזירה ידית לסמן. כאן מכיל הפרמטר hInst את הידית למודול המכיל את סמן העכבר, ואילו הפרמטר lpszName מכיל את שם סמן העכבר. כדי להשתמש באחד הסמנים ממלאי המערכת, עליך להקצות לפרמטר הראשון ערך NULL, ואילו לשני עליך להקצות שם של אחד מסמני המערכת, באמצעות שם המאקרו שלו. לפניך מספר מסמני המערכת הנפוצים ביותר.

שם המאקרו לסמן	תיאור
IDC_ARROW	בירירת המחדל - מצביע בצורת חץ
IDC_CROSS	צורת צלב-כוונת
IDC_IBEAM	צורת I
IDC_WAIT	צורת שעון חול

הוראת ההשמה hbrBackground שאחרי הוראות אלו יוצרת ידית למברשת צבע, ושני משפטי ההשמה הנותרים מציבים את ערכי ברירת המחדל של החלון עבור שם התפריט ושם מחלקת החלון.

לאחר שהגדרת בצורה מלאה מחלקה לחלון שלך, היא נרשמת בחלונות 9x באמצעות פונקציית API בשם ()RegisterClass, שתבניתה הטיפוסית מובאת להלן:

```
ATOM RegisterClass(CONST WNDCLASS *lpWClass);
```

הפונקציה מחזירה ערך המגדיר את סגנון החלון. ATOM הוא Typedef שפירושו WORD. לכל מחלקת חלונות מוקצה ערך ייחודי. lpWClass חייב להיות המען של מבנה WNDCLASS.

כיצד ליצור חלון

לאחר שהוגדר סגנון החלון (או המחלקה), יכול היישום שלך ליצור בפועל חלון מאותו סגנון באמצעות הפונקציה ()CreateWindow, שהגדרתה היא:

```
HWND CreateWindow(
  LPCSTR lpClassName, /* name of window class */
  LPCSTR lpWindowName, /* title of window */
  DWORD dwStyle, /* type of window */
  int X, int Y, /* upper-left coordinates */
  int nWidth, int nHeight, /* dimensions of window */
```

```
  HWND hWndParent, /* handle of parent window */
  HMENU hMenu, /* handle of main menu */
  HINSTANCE hInstance, /* handle of creator */
  LPVOID lpParam /* pointer to additional info */
);
```

כפי שעולה מעיון בתוכנית השלד, ניתן להשאיר רבים מהפרמטרים המועברים לפונקציה ()CreateWindow לפי ברירת המחדל, או להקצות להם ערך NULL. למעשה, במרבית המקרים הפרמטרים X, Y, Width ו-Height משתמשים במאקרו CW_USEDEFAULT, המורה לחלונות לבחור גודל ומיקום מתאימים לחלון, כאשר מדובר בחלונות שהסוג שלהם (dwStyle) מורכב גם מ-WS_OVERLAPPED (במידה וסוג החלון מורכב גם מ - WS_CHILD אין אפשרות להשתמש במאקרו זה, משום שהרוחב והאורך של החלון יהיו - 0). אם אין לחלון חלון-אב, כמו במקרה של תוכנית השלד, אזי יש להגדיר את הפרמטר hParent כ HWND_DESKTOP (תוכל להשתמש גם בערך NULL עבור פרמטר זה). אם החלון אינו מכיל תפריט ראשי, חובה להקצות ערך NULL גם לפרמטר hMenu (במקרה זה, החלון ישתמש בתפריט שנרשם במחלקת החלון). כמו כן, אם לא נחוץ מידע נוסף, כמו במרבית המקרים, יקבל גם הפרמטר lpParam ערך NULL (הסוג LPVOID מוגדר ב-typedef כ-*void. מבחינה היסטורית, **LPVOID** בא במקום "long pointer to **void**"). פרמטר זה הינו מצביע לנתונים שהפונקציה CreateWindow צריכה להציב בהודעה **WM_CREATE**. עבור חלונות בנים בממשק מרובה מסמכים (MDI), הערך lpParam צריך להיות מצביע למבנה CLIENTCREATESTRUCT. עבור רוב החלונות שאינם חלונות לקוח של MDI, עליך להציב NULL.

ארבעת הפרמטרים הנותרים יוגדרו על ידי התוכנית שלך. הראשון, lpszClassName, מצביע למחרוזת קבועה המסתיימת ב-NULL, שמכילה שם תקף של מחלקת חלון. אפשר ליצור שם זה בשני אופנים, על ידי התוכנית באמצעות השימוש ב-RegisterClass או על ידי ערך מובנה (ערך שמוגדר ונמצא במערכת) של סוג חלון. כותרת החלון היא מחרוזת שהמצביע שלה הוא הפרמטר lpszWinName. המחרוזת יכולה להיות ריקה, אבל בדרך כלל נהוג לתת כותרת לחלון. סגנון (או סוג) החלון שיווצר בפועל נקבע על ידי הערך של הפרמטר dwStyle, שמייצג את הסגנונות האפשריים של חלון. המאקרו **WS_OVERLAPPEDWINDOW** (בו השתמשנו), מגדיר חלון תקני המכיל תפריט מערכת, קו מתאר, לחצני הקטנה, הגדלה וסגירה. סגנון זה הוא הנפוץ ביותר, אך תוכל לעצב לעצמך חלון לפי הסגנון המתאים לך. כדי לעשות זאת, עליך להפעיל את האופרטור OR על פקודות המאקרו השונות של סגנונות שאתה מעוניין בהם. לפניך כמה מהסגנונות הנפוצים.

מאקרו לסגנונות	תיאור התכונה
WS_OVERLAPPED	חלון חופף עם קווי גבול
WS_MAXIMIZEBOX	לחצן הגדלה

מאקרו לסגנונות	תיאור התכונה
WS_MINIMIZEBOX	לחצן הקטנה
WS_SYSMENU	תפריט מערכת
WS_HSCROLL	פס גלילה אופקי
WS_VSCROLL	פס גלילה אנכי

קיימים הרבה ערכים נוספים. למידע מפורט חפש במדריכים המקוונים.

הפרמטר hThisInst חייב להכיל את הידית של המופע הנוכחי של היישום.

הפונקציה ()CreateWindow מחזירה את הידית של החלון שהיא יוצרת, או ערך NULL אם לא ניתן ליצור את החלון.

גם לאחר שנוצר, החלון אינו מוצג על המסך. כדי לגרום להצגתו, עליך להפעיל את הפונקציה ()ShowWindow, שהגדרתה היא :

```
BOOL ShowWindow(HWND hwnd, int nHow);
```

הפרמטר hwnd מכיל את הידית של החלון להצגה. מצב התצוגה נקבע בפרמטר nHow. בפעם הראשונה שהחלון מוצג על המסך, עליך להעביר את הפרמטר nWinMode בפונקציה ()WinMain כפרמטר nHow. בקריאות הבאות לפונקציה זו, אתה יכול להשתמש בערכים נוספים.

בתוכנית השלד כלולה קריאה לפונקציה ()UpdateWindow, למרות שמבחינה טכנית אין צורך בכך בתוכנית השלד, מפני שהקריאה לפונקציה זו נחוצה כמעט בכל יישום שתיצור לחלונות 9x.

למעשה, הפונקציה מורה לחלונות לשלוח הודעה ליישום שלך על כך שיש לעדכן את החלון הראשי .

לולאת ההודעות

הקטע האחרון של הפונקציה WinMain שבתוכנית **generic** עוסק בלולאת while, שמטפלת בהודעות המערכת. כל תוכנית Windows שתכתוב תשתמש ב**לולאת הודעות** (Message Loop). מטרתה בתוך הפונקציה לקבל ולעבד את ההודעות ששולחת חלונות. בעת ההרצה שלה, מקבל היישום הודעות ללא הרף. ההודעות נשמרות בתור ההודעות של היישום עד שניתן לקרוא ולעבד אותן.

הפורמט הרגיל של לולאת ההודעות מופיע להלן :

```
while(GetMessage(&msg, NULL, 0,0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
```

בכל פעם שהיישום שלך מוכן לקריאת הודעה נוספת, עליו להפעיל את הפונקציה ()GetMessage, שהגדרתה מובאת להלן:

```
BOOL GetMessage(lpMSG msg, HWND hWnd, UINT min, UINT max);
```

את הפונקציה GetMessage תכתוב, בדרך כלל, כמו בדוגמה זו:

```
BOOL GetMessage (&msg, NULL, 0,0);
```

הפונקציה GetMessage מחזירה ערך **אמת** (True) או **שקר** (False). היא מחזירה את הערך true עד שהיא מקבלת את ההודעה WM_QUIT (במקרה של שגיאה הפונקציה תחזיר 1– . למשל: אם הערך ב-HWND הוא ידית לחלון בלתי תקין). טבלה 2.2 מפרטת את הפרמטרים של הפונקציה GetMessage.

טבלה *2.2*: הפרמטרים של הפונקציה GetMessage.

שם הפרמטר	סוג	מטרה
lpMsg	MSG	החזרת מצביע למבנה מסוג MSG. לאחר טבלה זו תמצא את הגדרת המבנה MSG.
HWnd	HWND	ידית החלון שמקבל את ההודעות. בדרך כלל מציבים NULL עבור פרמטר זה, שמורה ל-GetMessage לקבל את כל ההודעות עבור המטלה הנוכחית.
UMsgFilterMin	UINT	ערך ההודעה המינימלי שצריך לקבל. בדרך כלל, מציבים 0 עבור פרמטר זה.
UMsgFilterMax	UINT	ערך ההודעה המקסימלי שצריך לקבל. אם תציב 0 עבור שני הפרמטרים uMsgFilterMin ו-uMsgFilterMax, הפונקציה GetMessage תקבל את כל ההודעות.

כמו שראית בטבלה 2.2, הפונקציה GetMessage מקבלת פרמטר מסוג MSG. Windows מגדירה את הסוג MSG בקובץ הכותר **winuser.h** כמו שתראה להלן:

```
typedef struct tagMSG {
    HWND              hwnd;     // window handle
    UINT              message;  // message ID
    UINT              wParam;   // wParam value
    LPARAM            lParam;   // lParam value
    DWORD             time;     // milliseconds since startup
    POINT             pt;// screen coordinates of
 } MSG;               current mouse location
```

כל אלמנט במבנה מסוג MSG חשוב, אבל האלמנט (או למעשה הפרמטר) שתשתמש בו לרוב הוא האיבר **message**, שמתייחס לרבות מהגדרות Windows לקבועי הודעות. תלמד יותר על קבועי הודעות בסעיפים שבהמשך ספר זה.

השדה hwnd שבמבנה MSG מכיל את הידית של החלון שאליו מיועדת ההודעה. כל ההודעות של Win32 הן מספרים שלמים ב- 32 סיביות, וההודעה נשמרת בשדה message. מידע נוסף הנוגע להודעות מועבר בפרמטרים wParam ו-lParam. הסוג WPARAM הוא הגדרת typedef ל-UINT, והסוג LPARAM הוא הגדרת typedef ל-LONG.

השעה שבה נשלחה ההודעה תצוין באלפיות השנייה בשדה time.

הרכיב (Member) pt יכיל את הקואורדינטות שבהן היה העכבר בעת שההודעה נשלחה. הקואורדינטות נשמרות במבנה POINT, שהגדרתו נראית כך:

```
typedef struct tagPOINT {
  LONG x, y;
} POINT;
```

אם אין הודעות בתור ההודעות של היישום, הקריאה לפונקציה ()GetMessage תחזיר את השליטה לחלונות (נושא ההודעות יידון ביתר פירוט בפרק הבא).

בתוך לולאת ההודעות, מופעלות שתי פונקציות. הראשונה בהן היא הפונקציה ()**TranslateMessage**. פונקציה זו מקבלת הודעת מקש וירטואלי (כמו VK_TAB) שהמערכת יוצרת כאשר המשתמש מקיש על מקש כלשהו, ומשגרת את ההודעה המתאימה WM_CHAR לתור ההודעות של היישום (WM_CHAR מייצגת עבור Windows את ההודעה על קבלת תו). אם ההודעה אינה של מקש וירטואלי, WM_CHAR מחזירה הודעת false ואינה מעבדת את ההודעה. אף שהדבר אינו נחוץ בכל יישום, מרבית היישומים מפעילים את הפונקציה ()TranslateMessage, מכיון שהיא נחוצה לאפשר שילוב מלא של המקלדת בתוכנית היישום.

לאחר שההודעה נקראה ותורגמה, היא נשלחת בחזרה לחלונות 9x באמצעות הפונקציה ()DispatchMessage. הפונקציה **DispatchMessage** קוראת לפונקציית המשוב (Callback Function) שהוגדרה ברישום מחלקת החלון, כדי שתטפל בהודעה זו. לרוב תשתמש בפונקציה DispatchMessage כפי שתראה בהצהרה שלהלן:

```
long DispatchMessage (CONST MSG* lpMsg);
```

למרות שהפונקציה DispatchMessage מחזירה ערך מסוג long, התוכניות שלך יתעלמו בדרך כלל מערך זה, מכיון שהשימוש בו בתוכנית שלך חסר משמעות.

✍ **הערה:** לולאת ההודעות **חייבת** להכיל את הפונקציה DispatchMessage, אחרת פונקציית החלון לא תקבל את ההודעות והתוכנית לא תוכל לטפל בהודעות שהמערכת שולחת.

עם סיום לולאת ההודעות, הפונקציה ()WinMain מסתיימת בכך שהיא שולחת לחלונות 9x את הערך שמכיל השדה msg.wParam. ערך זה הוא הקוד החוזר המופק כאשר התוכנית שלך מסתיימת.

פונקציית החלון

הפונקציה השנייה ביישום השלד היא פונקציית החלון. במקרה זה, נקראת הפונקציה ()WndProc, אולם תוכל להעניק לה כל שם שתבחר. פונקציית החלון מעבירה את ארבעת החברים הראשונים במבנה MSG כפרמטרים שלה.

פונקציית החלון של יישום השלד מגיבה בצורה פעילה רק על שתי הודעות:

- ✰ WM_DESTROY. הודעה זו נשלחת כאשר המשתמש מסיים את התוכנית. כשהיא מתקבלת, חייבת התוכנית שלך לבצע קריאה לפונקציה ()PostQuitMessage. הפרמטר לפונקציה זו הוא קוד יציאה המוחזר בתוך השדה msg.wParam, שבפונקציה ()WinMain. הקריאה לפונקציה ()PostQuitMessage גורמת למשלוח הודעת WM_QUIT אל היישום, דבר שבתורו גורם לפונקציה ()GetMessage להחזיר ערך false, וכך לעצור את התוכנית.
- ✰ WM_COMMAND. הודעה זו תישלח כאשר המשתמש ילחץ על התפריט. המשתמש יכול ללחוץ בתפריט בשני מקומות: Test, Exit. כאשר הוא ילחץ על Test יופעל IDM_TEST. המערכת לא תעשה כלום (זה רק הכנה לשימוש מאוחר יותר). כאשר המשתמש ילחץ על Exit, התוכנית תפעיל את הפונקציה DestroyWindow שבעצמה תסיים את התוכנית, ותשלח הודעה WM_DESTROY.

כל שאר ההודעות המתקבלות על ידי פונקציית החלון ()WndProc מועברות לחלונות באמצעות הפונקציה ()DefWindowProc, המכילה את ברירת המחדל לאופן העיבוד. צעד זה הוא צעד נחוץ, כיוון שחובה לטפל בכל הודעה בדרך זו או אחרת.

2.15 כללים לקביעת שמות

אם התכנות לחלונות 9x חדש עבורך, ייתכן שכמה משמות המשתנים והפרמטרים בתוכנית השלד ובתיאור הנלווה נראו לך מוזרים. הדבר נובע מקביעת שמות אלה על פי כללים מסוימים שהומצאו על ידי מיקרוסופט לצורך התכנות החלונאי. כשמדובר בפונקציות, על השם לכלול פועל ולאחריו שם עצם. האות הראשונה של הפועל ושם העצם יהיו רישיות. לקביעת שמות הפונקציות בספר זה, ישמשו אותנו ברוב המקרים, כללים אלה.

טבלה 2.3: התווים המשמשים כתחיליות לציון סוג המשתנה

התחילית	תיאור הנתון
b	בוליאני (סיבית אחת)
c	תו (סיבית אחת)
dw	מספר שלם ארוך ובלתי מסומן
f	שדה-סיביות בן 16 סיביות (דגלים)
fn	פונקציה
h	ידית
l	מספר שלם ארוך
lp	מצביע ארוך
n	מספר שלם קצר
p	מצביע
pt	מספר שלם ארוך המכיל קואורדינטות של מסך
w	מספר שלם קצר ובלתי מסומן
sz	מצביע על מחרוזת שסיומה הוא ב-NULL
lpsz	מצביע ארוך, המצביע על מחרוזת שסיומה הוא ב-NULL
rgb	מספר שלם ארוך המכיל ערכי צבע RGB

אשר לשמות של משתנים, מיקרוסופט בחרה להשתמש בשיטה מסובכת למדי של שתילת סוג הנתון בשם. כדי לעשות זאת, מוסיפים תחילית (Prefix) באותיות קטנות לתחילת שם המשתנה, וזו מציינת את סוג המשתנה. השם עצמו מתחיל באות רישית. תחיליות הסוג מפורטות בטבלה 2.3. למען האמת, השימוש בתחיליות אלו שנוי במחלוקת, ואינו מקובל על הכל. מתכנתי חלונות רבים משתמשים בשיטה זו, אך יש רבים אחרים שאינם משתמשים בה. בספר זה השתמשנו בשיטת מיקרוסופט ככל שהדבר התבקש ונראה הגיוני, אולם אין זה מחייב אותך כלל. באפשרותך להשתמש בכל מערכת כללים לקביעת שמות שתרצה.

פרק 3

עיבוד הודעות

כפי שהסברנו בפרק 2, חלונות מתקשרת עם יישומים על ידי משלוח הודעות. לכן, עיבוד ההודעות האלו הוא בעל חשיבות מכרעת בכל יישומי חלונות 9x. בפרק הקודם למדת כיצד ליצור יישום שלד לחלונות 9x. בפרק זה, נרחיב את השלד, כך שיוכל לקבל ולעבד כמה מההודעות הנפוצות ביותר.

3.1 מהן הודעות?

חלונות 9x מפיקה מיגוון רחב של הודעות. כל הודעה מיוצגת על ידי ערך שהוא מספר שלם ייחודי, בן 32 סיביות. קובץ הכותר windows.h מכיל שמות תקניים להודעות אלו. כשתידרש להתייחס להודעה, תשתמש בדרך כלל בשם המאקרו של כל הודעה, ולא בערך עצמו. להלן שמות המאקרו של כמה מההודעות הנפוצות ביותר של חלונות:

```
WM_CHAR
WM_PAINT
WM_MOVE
WM_CLOSE
WM_LBUTTONUP
WM_LBUTTONDOWN
WM_COMMAND
```

לכל הודעה נלווים שני ערכים נוספים, אשר מכילים מידע הנוגע להודעה שאליה הם קשורים. אחד הערכים הוא מסוג WPARAM, והשני הוא מסוג LPARAM. חלונות שני הערכים האלה מתורגמים למספרים שלמים בני 32 סיביות. השמות המקובלים לערכים אלה הם wParam ו-lParam, בהתאמה. בדרך כלל, שמורים בהם פרטים כגון קואורדינטות לסמן או לעכבר, ערך של לחיצה על מקש, או ערכים הקשורים במערכת כגון גודל התווים. בהמשך נברר את משמעות הערכים השמורים בפרמטרים wParam ו-lParam לגבי כל הודעה שנעסוק בה.

כפי שהזכרנו בפרק 2, הפונקציה שמעבדת בפועל את ההודעות היא פונקציית החלון של התוכנית. בוודאי תזכור שפונקציה זאת מקבלת ארבעה פרמטרים: הידית של החלון שההודעה מיועדת לו, ההודעה עצמה, ולבסוף הפרמטרים wParam ו-lParam.

לפעמים קורה ששתי פיסות מידע מקודדות אל תוך המילים המרכיבות את הפרמטרים wParam ו-lParam. כדי לאפשר גישה נוחה לכל אחד מהחצאים של פרמטרים אלה, חלונות מגדירה שתי פקודות מאקרו, ששמותיהן LOWORD ו-HIWORD. פקודות אלו מחזירות את **מילת הסדר-הגבוה** ואת **מילת-הסדר הנמוך** של מספר שלם ארוך, בהתאמה. השימוש בהם הוא כזה:

```
x = LOWORD(lParam);
x = HIWORD(lParam);
```

בהמשך תראה את פקודות המאקרו האלו בפעולה.

למדת שתוכניות מגיבות לקלט מהמשתמש בפורמט של הודעות מערכת. בסעיפים קודמים התוכניות בדקו את ההודעה WM_COMMAND כדי לדעת אם המשתמש שלח פקודה לתוכנית, ואם צריך להגיב לבחירת המשתמש מתפריט או מתוך פקד שיצרו את ההודעה. כאשר התוכנית פועלת באמצעות פעולות עכבר, חובה עליה לבדוק את הודעות Windows שמפורטות בטבלה 3.1 שלהלן.

הודעות Windows בתגובה לפעולות עכבר.

טבלה 3.1: הודעות Windows בתגובה לפעולות עכבר.

הודעה	פירוש
WM_CAPTURECHANGED	Windows שולחת את ההודעה הזו לחלון שאיבד את **לכידת העכבר** (Mouse Capture).
WM_LBUTTONDBLCLK	המשתמש לחץ על לחצן העכבר השמאלי פעמיים.
WM_LBUTTONDOWN	המשתמש לחץ על לחצן העכבר השמאלי, והלחצן עדיין לחוץ.
WM_LBUTTONUP	המשתמש שחרר את לחצן העכבר השמאלי.
WM_MBUTTONDBLCLK	המשתמש לחץ על לחצן העכבר האמצעי פעמיים (רק בעכברים בעלי שלושה לחצנים).
WM_MBUTTONDOWN	המשתמש לחץ על לחצן העכבר האמצעי, והלחצן עדיין לחוץ (רק בעכברים בעלי שלושה לחצנים).
WM_MBUTTONUP	המשתמש שחרר את לחצן העכבר האמצעי (רק בעכברים בעלי שלושה לחצנים).
WM_MOUSEACTIVATE	המשתמש לחץ בעכבר בחלון שאינו פעיל כרגע.

הודעה	פירוש
WM_MOUSEMOVE	המשתמש גרר את העכבר בחלון.
WM_NCLBUTTONDBLCLK	המשתמש לחץ על לחצן העכבר השמאלי פעמיים בחלון, בשטח שאינו שטח הלקוח (Non-Client Area).
WM_NCLBUTTONDOWN	המשתמש לחץ על לחצן העכבר השמאלי בחלון, בשטח שאינו שטח הלקוח, והלחצן עדיין לחוץ.
WM_NCLBUTTONUP	המשתמש שחרר את לחצן העכבר השמאלי בשטח שאינו שטח הלקוח של החלון.
WM_NCMBUTTONDBLCLK	המשתמש לחץ על לחצן העכבר האמצעי פעמיים בשטח שאינו שטח הלקוח בחלון (רק בעכבר בעל שלושה לחצנים).
WM_NCMBUTTONDOWN	המשתמש לחץ על לחצן העכבר האמצעי בשטח שאינו שטח הלקוח בחלון, והלחצן עדיין לחוץ (רק בעכבר בעל שלושה לחצנים).
WM_NCMBUTTONUP	המשתמש שחרר את לחצן העכבר האמצעי בשטח שאינו שטח הלקוח של החלון (רק בעכבר בעל שלושה לחצנים).
WM_NCMOUSEMOVE	המשתמש גרר את העכבר בחלון, בשטח שאינו שטח הלקוח.
WM_NCRBUTTONDBLCLK	המשתמש לחץ על לחצן העכבר הימני פעמיים בשטח שאינו שטח הלקוח בחלון.
WM_NCRBUTTONDOWN	המשתמש לחץ על לחצן העכבר הימני בשטח שאינו שטח הלקוח בחלון, והלחצן עדיין לחוץ.
WM_NCRBUTTONUP	המשתמש שחרר את לחצן העכבר הימני בשטח שאינו שטח הלקוח בחלון.
WM_RBUTTONDBLCLK	המשתמש לחץ על לחצן העכבר הימני פעמיים.
WM_RBUTTONDOWN	המשתמש לחץ על לחצן העכבר הימני, והלחצן עדיין לחוץ.
WM_RBUTTONUP	המשתמש שחרר את לחצן העכבר הימני.

כאשר יישום מקבל הודעה מהעכבר, הערך lParam מכיל את קואורדינטות X ו-Y של סמן העכבר שעל המסך. Windows מאחסנת את הקואורדינטה Y במילת הסדר-הגבוה של lParam, ואת הקואורדינטה X - במילת הסדר-הנמוך. צריך להשתמש במאקרוס LOWORD ו-HIWORD כדי להפריד בין שני הערכים, ולטפל בכל אחד מהם בנפרד.

הסעיף הבא בוחן את ההודעה WM_MOUSEMOVE כדוגמה להודעת עכבר של Windows, וכולל קטע קוד שמראה איך שולפים את ערכי הקואורדינטות מהפרמטר lParam.

✍ **הערה:** במערכות חלונות מתקדמות יותר מ- Windows 95 קיימות הודעות נוספות המתקבלות מהעכבר. לפירוט נוסף עיין במדריכים המקוונים של מיקרוסופט.

3.2 ההודעה WM_MOUSEMOVE

למדת בסעיף הקודם ש-Windows מעבירה ליישומים הודעות שונות כתגובה לפעולות בעכבר. אחת ההודעות הנפוצות ביותר היא ההודעה WM_MOUSEMOVE. Windows שולחת את ההודעה הזו לחלון בכל פעם שהסמן זז. אם אין חלון שכבר לכד את מידע העכבר, Windows שולחת את ההודעה WM_MOUSEMOVE לחלון שבו נמצא הסמן. אחרת, היא שולחת את ההודעה אל החלון שלכד את העכבר. כשהתוכנית מקבלת את ההודעה WM_MOUSEMOVE, היא צריכה לבדוק את מספר ערכים, לפני שהיא מגיבה להודעה:

```
fwKeys = wParam;        // key flags
xPos = LOWORD(lParam); // horizontal position of cursor
yPos = HIWORD(lParam); // vertical position of cursor
);
```

למדת בסעיף הקודם שהערך lParam מכיל את קואורדינטות המיקום X ו-Y של סמן העכבר. הפרמטר wParam מכיל את הערך fwKeys, אשר מציין אם מקשים וירטואליים כלשהם לחוצים. הערך fwKeys יכול להיות כל צירוף של הערכים המפורטים בטבלה 3.2.

טבלה 3.2: הערכים האפשריים של הפרמטר fwKeys.

ערך	תיאור
MK_CONTROL	מקש Ctrl לחוץ.
MK_LBUTTON	הלחצן השמאלי של העכבר לחוץ.
MK_MBUTTON	הלחצן האמצעי של העכבר לחוץ.
MK_RBUTTON	הלחצן הימני של העכבר לחוץ.
MK_SHIFT	המקש Shift לחוץ.

כדי ללכוד את תנועות העכבר מתוך פונקציית ההודעות, על התוכנית להשתמש בקוד דומה לזה שמוצג בקטע הבא:

```
  case WM_MOUSEMOVE :
         {
           nXPos = LOWORD(lParam);
           nYPos = HIWORD(lParam);
           // other statments
```

בקטע קוד זה, התוכנית מציבה את המיקום הנוכחי של העכבר במשתנים nXpos ו-nYpos בכל פעם שהמשתמש מזיז את העכבר.

3.3 קריאת לחצני העכבר

בסעיף 3.2 למדת להשתמש בהודעה WM_MOUSEMOVE שנשלחת לחלון, כדי לאפשר לתוכניות להגיב לתנועות סמן העכבר על המסך. באופן דומה, הוספת קוד לתוכניות כדי שיגיבו לפעולות לחצני העכבר, גם היא דבר פשוט. לדוגמה, כשרוצים שהתוכנית תגיב ללחיצה כפולה של המשתמש בעכבר, בנקודה כלשהי שנמצאת בשטח הלקוח של החלון, התוכנית תשתמש בקוד שנמצא בפונקציה WndProc ודומה לקטע הקוד הזה:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
  switch (uMsg)
   {
   case WM_LBUTTONDBLCLK :
      MessageBox(hWnd, "Button Double-Clicked", NULL, MB_OK);
      break;
```

אולם, כשרוצים שתגובת התוכנית תהיה שונה כאשר המשתמש מחזיק את המקש Ctrl לחוץ בזמן הלחיצה על העכבר, אפשר לכתוב את קטע הקוד הזה:

```
case WM_LBUTTONDBLCLK :
   if ((wParam & MK_CONTROL) == MK_CONTROL)
    {
        MessageBox(hWnd, "Button Double-Clicked with
                 Control Key Down", NULL, MB_OK);
        break;
    }
   else
    {
        MessageBox(hWnd, "Button Double-Clicked without
                 Control Key Down", NULL, MB_OK);
        break;
     }
```

אפשר לראות מהקוד שהתוכנית יכולה לבדוק אם המשתמש לוחץ כרגע על **מקש וירטואלי** (Virtual Key) נתון. היא עושה זאת על ידי ביצוע AND הפועל על סיביות (Bitwise AND), בין ערך הפרמטר wParam לבין ערך המקש הווירטואלי שרוצים לבדוק. ככל שהתוכניות שלך יילכו ויהיו יותר מורכבות, תרבה לבדוק את המקשים הווירטואליים בעת הטיפול באירועי העכבר. ראוי לזכור שלמרות ש-Windows מוסרת רק מספר מסוים של מקשים וירטואליים ב-wParam בהקשר ללחיצות העכבר, היא תומכת במספר גדול של מקשים וירטואליים, כפי שתלמד בסעיף 3.5.

3.4 תגובה לאירועי מקלדת

למדת ש-Windows שולחת הודעות ליישום בכל פעם שהמשתמש מבצע פעילות כלשהי בעכבר. כך היא גם שולחת הודעות ליישום בכל פעם שהמשתמש מקיש במקלדת. ההודעה ש-Windows תשלח תהיה אחת מאלו שמפורטות בטבלה 3.3.

טבלה 3.3: הודעות המערכת ש-Windows שולחת בעת פעולות מקלדת.

הודעה	תיאור
WM_ACTIVATE	הקשה בתוך חלון שאינו פעיל.
WM_CHAR	קוד ASCII של האות, אם המשתמש הקיש על מקש תו.
WM_GETHOTKEY	מאפשר לתוכניות לקלוט מקש חם (Hot Key, או מקש מהיר) שהוגדר קודם לכן עבור החלון.
WM_HOTKEY	המשתמש הקיש על מקש חם.
WM_KEYDOWN	המשתמש הקיש על מקש.
WM_KEYUP	המשתמש שחרר את המקש.
WM_KILLFOCUS	ההודעה נשלחת לחלון מייד לפני שהוא מאבד את מיקוד הקלט מהמקלדת.
WM_SETFOCUS	ההודעה נשלחת לחלון מייד לאחר שהחלון מקבל את מיקוד הקלט מהמקלדת.
WM_SETHOTKEY	מאפשר לתוכניות לקבוע "מקש חם" עבור חלון.
WM_SYSCHAR	קוד ASCII של האות שהמשתמש הקיש בעת שמקש Alt לחוץ.
WM_SYSKEYDOWN	המשתמש הקיש על מקש כאשר מקש Alt לחוץ.
WM_SYSKEYUP	המשתמש שחרר את המקש וגם את המקש Alt.

הפונקציה TranslateMessage שבלולאת ההודעות של המטלה יוצרת את ההודעה WM_CHAR אם היא מצליחה לפענח את התו כתו ASCII. ככלל, היישום משתמש בהודעה WM_KEYDOWN כדי לבדוק את **מקשי הפונקציות** (Function Keys), **מקשי סמן** (Cursor Keys), **המקלדת הנומרית** (Numeric Keypad) ו**מקשי העריכה** (Edit Keys), כמו PageUp ו-PageDown. מקשים אלה מנצלים בצורה הטובה ביותר את קוד המקשים הווירטואליים (Virtual-Key Codes), עליו נלמד בסעיף הבא.

עם זאת, על התוכניות להשתמש ב-WM_CHAR כדי לקבל **אותיות** (Characters), **סְפָרוֹת** (Numbers) ו**סימנים שניתנים בהדפסה** (Printable Symbols). השימוש בהודעה WM_CHAR כדי לטפל בהקשות אלו טוב יותר, מפני ש-ASCII מציבה ערכים שונים עבור אותיות רגילות (Lowercase Letters) ואותיות רישיות (Capital Letters). עם ההודעה WM_KEYDOWN, היישום חייב לבדוק את התו שהקיש המשתמש וגם את מצב מקש Shift. ההודעה WM_CHAR מטפלת בהקשת Shift.

כשהמשתמש מקיש על מקש Alt ובעודו לחוץ - הוא מקיש על מקש נוסף, היישום מקבל את רצף ההודעות הבאות: WM_SYSKEYDOWN, WM_SYSCHAR, WM_SYSKEYUP. התוכנית צריכה להשתמש בהודעות אלו כדי לבדוק רצף הקשות הנלוות למקש ALT.

3.5 מקשים וירטואליים (Virtual Keys)

למדת בסעיף הקודם שכדי לטפל בהקשה כתו ASCII, התוכנית צריכה להשתמש בהודעה WM_CHAR.

לפעמים, התוכנית עשויה לדרוש, או שתרשה למשתמש, להשתמש במקשים אחרים בנוסף למקשי ASCII הרגילים, כמו לדוגמה, מקשי החיצים, מקשי המספרים במקלדת הנומרית, וכדומה. כאשר משתמשים במקשים כאלה, צריך להשתמש בערכים של מקשים וירטואליים. השימוש במקשים וירטואליים משחרר אותך מלהתחשב בסוג המקלדת של המשתמש, מכיון שהקוד הווירטואלי של מקש הפונקציה הראשון צריך להיות תמיד זהה, ללא תלות ביצרן המקלדת או בסוג שלה. ההודעות WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN ו-WM_SYSKEYUP שולחות את קוד המקשים הווירטואליים בפרמטר wParam של ההודעה.

במכלול המבנה של קוד המקשים הווירטואליים, למספרים שבמקלדת הנומרית יש קוד מקשים וירטואלי נפרד. בנוסף, הקוד הווירטואלי עבור התווים ומקשי הספרות שקול לערך ASCII של האות הרישית (Uppercase). במילים אחרות, ל-a ו-A יש את הקוד VK_A. לבסוף, שים לב ששני מקשי Shift (הימני והשמאלי) יוצרים אותו קוד וירטואלי. טבלה 3.4 מפרטת את הקודים הווירטואליים של Win 32 API.

טבלה 3.4: קודי המקשים הווירטואליים של Win 32 API.

קוד המקש	ערך (הקסדצימלי)	השקול במקלדת או העכבר
VK_LBUTTON	01	לחצן שמאלי של העכבר.
VK_RBUTTON	02	לחצן ימני של העכבר.
VK_CANCEL	03	הטיפול ב-Control-Break.
VK_MBUTTON	04	לחצן אמצעי של העכבר, בעכבר עם שלושה לחצנים, או שני הלחצנים השמאלי והימני בו-זמנית.
VK_BACK	08	מקש מסג (Backspace).
VK_TAB	09	טבלר (Tab).
VK_CLEAR	0C	Clear.
VK_RETURN	0D	Enter.
VK_SHIFT	10	Shift.
VK_CONTROL	11	Ctrl.
VK_MENU	12	Alt.
VK_PAUSE	13	Pause.
VK_CAPITAL	14	Caps Lock.
VK_ESCAPE	18	Esc.
VK_SPACE	20	Space Bar.
VK_PRIOR	21	Page Up.
VK_NEXT	22	Page Down.
VK_END	23	End.
VK_HOME	24	Home.
VK_LEFT	25	Left Arrow.
VK_UP	26	Up Arrow.
VK_RIGHT	27	Right Arrow.
VK_DOWN	28	Down Arrow.

השקול במקלדת או העכבר	ערך (הקסדצימלי)	קוד המקש
Select	29	VK_SELECT
Execute	2B	VK_EXECUTE
Print Screen	2C	VK_SNAPSHOT
Insert	2D	VK_INSERT
Delete	2E	VK_DELETE
Help	2F	VK_HELP
0 - 9	30-39	VK_0 - VK_9
A - Z	41-5A	VK_A - VK_Z
Numeric Keyboard0	60	VK_NUMPAD0
Numeric Keyboard1	61	VK_NUMPAD1
Numeric Keyboard2	62	VK_NUMPAD2
Numeric Keyboard3	63	VK_NUMPAD3
Numeric Keyboard4	64	VK_NUMPAD4
Numeric Keyboard5	65	VK_NUMPAD5
Numeric Keyboard6	66	VK_NUMPAD6
Numeric Keyboard7	67	VK_NUMPAD7
Numeric Keyboard8	68	VK_NUMPAD8
Numeric Keyboard9	69	VK_NUMPAD9
Multiply Key	6A	VK_MULTIPLY
Add Key	6B	VK_ADD
Separator Key	6C	VK_SEPARATOR
Subtract Key	6D	VK_SUBTRACT
Decimal Key	6E	VK_DECIMAL
Divide Key	6F	VK_DIVIDE
F1 - F24 Function Keys	70-87	VK_F1 - VK_F24

קוד המקש	ערך (הקסדצימלי)	השקול במקלדת או העכבר
VK_NUMLOCK	90	NumLock
VK_SCROLL	91	ScrollLock

3.6 הצגת המקשים הווירטואליים

למדת שהתוכנית מקבלת מ-Windows בדרך כלל אחד משלושה סוגי קלט מקלדת: קלט תווי ASCII **מוכרים**, המועברים במשתנה wParam של ההודעה WM_CHAR; קלט תווים **לא מוכרים** (Unrecognized), שאינם תווי מערכת (Non-System Character) ומועברים ב-wParam של ההודעה WM_KEYDOWN; וקלט **תווי מערכת** (System Character) המועברים ב-wParam של ההודעה WM_SYSKEYDOWN. גם למדת כיצד יכולה התוכנית לטפל בקלות בהודעות שמתקבלות מהעכבר. הטיפול בהודעות המקלדת קל ופשוט. לדוגמה, קטע הקוד הבא מציג תווים מוכרים בתוך תיבת עריכה של טקסט, ואם התו שאינו מוכר הוא מקש פונקציה, הקוד מציג תיבת הודעה שמציינת זאת:

```
LRESULT CALLBACK DlgTestProc(HWND hWnd, UINT uMsg,
                             WPARAM wParam, LPARAM lParam)
{
  switch (uMsg)
   {
      case WM_CHAR :
         GetDlgItemInt(hWnd, IDC_EDITBOX);
         Insert(szEditBox, wParam);
         SetDlgItemText(hWnd, IDC_EDITBOX, szEditBox);
      break;
      case WM_KEYDOWN:
         switch (wParam)
          {
            case VK_F1:
              MessageBox(hWnd, "Function 1 Key Pressed",
                         NULL, MB_OK);
              break;
            case VK_F2:
              MessageBox(hWnd, "Function 2 Key Pressed",
                         NULL, MB_OK);
              break;
          // More Virtual key tests here
          }
    }
}
```

אפשר לראות שהתוכנית בודקת את הערכים הווירטואליים של מקשי הפונקציות ש-Windows מאחסנת בפרמטר wParam, כדי לקבוע את המקש הווירטואלי שהמשתמש הקיש.

3.7 מבט נוסף על השימוש בהודעה WM_KEYDOWN

Windows שולחת את ההודעה WM_KEYDOWN לחלון שנמצא במיקוד הקלט של המקלדת, כאשר המשתמש מקיש על **מקש שאינו מקש מערכת** (Non-System Key). מקש שאינו מקש מערכת הוא מקש שהמשתמש מקיש עליו ללא שילוב עם Alt. בסעיף 3.5 ראית ש-Windows מעבירה בתוך הפרמטר wParam את קוד המקש הווירטואלי שהמשתמש הקיש. אך היא גם מעבירה בפרמטר lParam את הערך lKeyData. הפרמטר lKeyData מגדיר **מספר החזרות** (Repeat Count), **קוד סריקה** (Scan Code), **דגל מקש-מורחב** (Extended-Key Flag), **קוד הקשר** (Context Code), **דגל מצב-מקש קודם** (Previous Key-State Flag) **ודגל מצב-מעבר** (Transition-State Flag), כמו שתוכל לראות בטבלה 3.5.

טבלה 3.5: הערכים ש-Windows מעבירה בתוך הפרמטר lParam עם ההודעה WM_KEYDOWN.

סיבית	תיאור
0 - 15	מספר החזרות. ערך זה הוא מספר ההקשות החוזרות כתוצאה מהחזקת המקש לחוץ.
16 - 23	קוד הסריקה. הערך תלוי ביצרן הציוד (OEM - Original Equipment Manufacturer).
24	מקש מורחב (Extended Key), כמו המקשים Alt ו-Ctrl הימניים שנמצאים במקלדת המשופרת של 101 או 102 מקשים. כאשר ערך סיבית זו הוא 1, המקש הוא מקש מורחב; אחרת, המקש רגיל (שמאלי).
25 - 28	שמור, אינו בשימוש.
29	קוד הקשר (Context Code). הערך תמיד 0 עבור ההודעה WM_KEYDOWN.
30	המצב הקודם של המקש. סיבית זו נקבעת ל-1 אם המקש נלחץ לפני שליחת ההודעה; וערכה 0 אם המקש אינו לחוץ.
31	מצב מעבר. סיבית זו לעולם אינה נקבעת עבור ההודעה WM_KEYDOWN.

כשהמשתמש מקיש על מקש הפונקציה F10, הפונקציה DefWindowProc קובעת דגל פנימי. כאשר DefWindowProc מקבלת את ההודעה WM_KEYUP, הפונקציה בודקת אם הדגל הפנימי נקבע; אם כן, היא שולחת הודעה WM_SYSCOMMAND לחלון הראשי (Top-Level Window). DefWindowProc קובעת את ערך הפרמטר wParam של ההודעה ל-SC_KEYMENU.

תכונת החזרה האוטומטית (auto-repeat feature) שיש ל-Windows מאפשרת לה לשלוח יותר מהודעה WM_KEYDOWN אחת אל היישום, לפני שהיא שולחת את ההודעה WM_KEYUP. באפשרות התוכניות להשתמש במצב-קודם של המקש (סיבית 30) כדי לקבוע אם ההודעה WM_KEYDOWN מציינת את מצב המעבר הראשון, או את מצב המעבר החוזר.

במקלדות המשופרות בנות 101 ו-102 מקשים, המקשים המורחבים הם מקשי Alt ו-Ctrl הימניים בחלק המרכזי של המקלדת; המקשים Ins, Del, Home, End, Page Up, Page Down, ומקשי החיצים בחלק שמשמאל למקלדת הנומרית, ומקש הלוכסן (/) ו-Enter במקלדת הנומרית. ייתכן שמקלדות אחרות יתמכו בסיבית ההרחבה שבפרמטר lKeyData.

ייתכנו מצבים בהם תרצה לקבל מחרוזת שמייצגת את שם המקש. בסעיף הקודם טיפלנו במקשים ידועים בתוך הוראת switch. אולם ממשק Win 32 API מכיל את הרשימה של המקשים שיש להם קוד מקש וירטואלי, ובאפשרותך להשתמש בפונקציה GetKeyNameText כדי לקבל אותה. הפונקציה GetKeyNameText מקבלת מחרוזת שמייצגת את שם המקש. משתמשים בפונקציה GetKeyNameText כמו בדוגמה זו:

```
int GetKeyNameText(
 LONG lParam,           // second parameter of keyboard message
 LPTSTR lpString,       // address of buffer for key name
 int nSize              // maximum length of key name string
length
);
```

הפרמטר lpString מצביע למאגר שמקבל את שם המקש. הפרמטר nSize מגדיר את האורך המקסימלי, בתווים של שם המקש, כולל את התו המסיים NULL (הפרמטר nSize צריך להיות שווה לגודל המאגר שמוצבע על ידי הפרמטר lpString). הפרמטר wParam מגדיר את הפרמטר השני של הודעת המקלדת (כמו WM_KEYDOWN) שהפונקציה GetKeyNameText עומדת לטפל בו. החלקים של wParam שהפונקציה מפענחת מפורטים בטבלה 3.6.

טבלה 3.6: הסיביות של wParam ש-GetKeyNameText מפענחת.

סיביות	פירוש
16 - 23	קוד הסריקה (Scan Code).
24	דגל מקש מורחב (Extended-Key Flag), כדי להגדיר מספר מקשים במקלדת המורחבת.
25	סיבית "אל תתחשב" ("Don't care"). היישום שקורא לפונקציה GetKeyNameText קובע את הסיבית הזו, כדי לציין שהיא לא צריכה להבחין בין Ctrl שמאלי לבין ימני, או בין Shift שמאלי לבין ימני, וכדומה.

הפורמט של המחרוזות של שם המקש תלוי בפריסת המקלדת (Keybord Layout) הנוכחית. מנהל ההתקן של המקלדת מחזיק רשימת שמות בפורמט של מחרוזות תווים, עבור מקשים שהשם שלהם גדול מתו אחד. מנהל ההתקן של המקלדת מתרגם את שמות המקשים לפי פריסת המקלדת המותקנת. שם של מקש תו הוא התו עצמו. התקליטור שמצורף לספר זה מכיל את התוכנית **Show_Keys** (בתיקיה Books\59285\chap03\show_keys), אשר מציגה שם של מקש בכל פעם שהמשתמש מקיש על מקש כלשהו.

ב-case של ההודעה WM_KEYDOWN, התוכנית מטפלת ב**הקשרי התקן** (Device Contexts), עליהם תלמד בהמשך. הנקודה החשובה שסעיף זה רוצה לציין היא הקריאה לפונקציה GetKeyNameText, כפי שמוצג להלן:

```
GetKeyNameText(lParam, szName, 30);
```

התוכנית קוראת ל-GetKeyNameText, ואחר כך מוציאה את הפלט אל המאגר szName של החלון.

3.8 משך זמן הלחיצה הכפולה בעכבר

התוכניות מבצעות פעולות עיבוד שונות בעת קבלת קלט מהמשתמש. פעולה נפוצה של משתמש בתוכנית היא **לחיצה כפולה** (Double-Click) בעכבר, כאשר הסמן מוצב על אפשרות כלשהי בתפריט או שהוא מוצב על סמל כלשהו. לחיצה כפולה היא סדרה של שתי לחיצות בלחצן העכבר, כאשר השנייה מתרחשת פרק זמן מסוים לאחר הראשונה. אנו רוצים לשלוט בפרק הזמן שחולף עד הלחיצה השנייה, כדי ששתי הלחיצות תחשבנה לחיצה כפולה, ולא שתי לחיצות עוקבות. באפשרות התוכניות להשתמש בפונקציה SetDoubleClickTime כדי לשלוט במהירות הלחיצה הכפולה של העכבר. הפונקציה SetDoubleClickTime קובעת את משך הלחיצה הכפולה בעכבר. משך הלחיצה הכפולה הוא המספר המקסימלי של מילישניות שחולפות בין הלחיצה הראשונה לבין הלחיצה השנייה, ה"כפולה".

משתמשים בפונקציה SetDoubleClickTime בתוכניות, כמו שרואים בהגדרה שלהלן:

```
BOOL SetDoubleClickTime(
  UINT uInterval              // double-click interval
);
```

הפרמטר uInterval מגדיר את מספר המילישניות המקסימלי המותר בין הלחיצה הראשונה לבין הלחיצה השנייה, כדי ששתיהן תיחשבנה לחיצה כפולה. אם נקבע לפרמטר זה הערך אפס, Windows תשתמש בזמן ברירת המחדל של הלחיצה הכפולה, אשר שווה 500 מילישניות.

✍ **הערה:** הפונקציה SetDoubleClickTime משנה את משך הלחיצה הכפולה לכל החלונות שבמערכת. לכן, אם תשנה את משך הזמן הזה בתוכנית שלך, עליך **להחזיר** אותו לערכו הקודם לאחר שהתוכנית מסתיימת.

כמו שתוכל לקבוע את משך הזמן בין שתי לחיצות עכבר שמזוהות כלחיצה כפולה, תוכל גם לקבל את משך הלחיצה הכפולה הנוכחי בעכבר, באמצעות הפונקציה GetDoubleClickTime. ראינו שלחיצה כפולה היא שתי לחיצות עכבר רצופות בפרק זמן שאינו עולה על ערך שנקבע מראש. זהו המספר המקסימלי של מילישניות שחולפות בין הלחיצה הראשונה לבין הלחיצה העוקבת לה. משתמשים בפונקציה GetDoubleClickTime בתוכניות כמו בהגדרה שלהלן:

```
UINT GetDoubleClickTime(void);
```

אם הפונקציה GetDoubleClickTime מצליחה, הערך המוחזר של הפונקציה מגדיר את משך הלחיצה הכפולה במילישניות. כדי להבין טוב יותר את הטיפול שמבצעות הפונקציות SetDoubleClickTime ו-GetDoubleClickTime, התבונן בתוכנית **Get_Set_Dbl_Click**, שבתקליטור המצורף לספר זה (chap03\Get_Set_Dbl_Click).

בכל פעם שהמשתמש לוחץ לחיצה כפולה בעכבר, התוכנית מציגה תיבת הודעה שמציינת שהיא קיבלה את הלחיצה הכפולה. אך בכל פעם שהמשתמש בוחר את האפשרות !Test מהתפריט, התוכנית מגדילה את משך הלחיצה הכפולה בעשירית השנייה. שים לב, שכאשר התוכנית מסתיימת, היא מחזירה את משך הלחיצה הכפולה לערך המקורי שהיה לפני השינוי.

3.9 החלפת לחצני העכבר

ראית בסעיף 3.8, שיכולים להיות מצבי עבודה שונים שבהם צריך לשנות דברים שנקבעו מראש. אחד מאלה היה שינוי משך ההשהיה בין לחיצות העכבר, כדי שתיחשבנה לחיצה כפולה. כך גם ייתכן שצריך יהיה להחליף את לחצני העכבר ולהשתמש בלחצן הימני לביצוע פעולות שכרגיל נעשות על ידי הלחצן השמאלי, ולהיפך, להשתמש בלחצן השמאלי כדי לבצע את הפעולות שנעשות בדרך כלל על ידי הלחצן הימני. הפונקציה SawpMouseButton הופכת, או משחזרת, את פעולת לחצני העכבר השמאלי והימני. Windows מאפשרת את החלפת הלחצנים כדי להקל על מי שמשתמשים בעכבר ביד שמאל. בדרך כלל, רק **לוח הבקרה** (Control Panel) קורא לפונקציה SawpMouseButton, למרות שאפשר לקרוא לפונקציה באופן חופשי בתוכניות. כשחייבים להשתמש בתוכניות בפונקציה SawpMouseButton, צריך לקרוא לה כמו בהגדרה שלהלן:

```
BOOL SwapMousBoutton(
  BOOL fSwap            // reverse or restore buttons
);
```

הפרמטר fSwap מציין את התפקיד הנוכחי של לחצני העכבר: רגיל או הפוך. אם fSwap שווה True, לחצן העכבר השמאלי מייצר הודעות לחצן ימני ולחצן העכבר הימני מייצר הודעות לחצן שמאלי. אם fSwap הוא False, הלחצנים נמצאים בהגדרתם המקורית. בתקליטור שמצורף לספר זה תמצא את התוכנית **Swap_B** (בתיקיה chap03), שמחליפה את המצב של לחצני העכבר ומשחזרת אותם ולהיפך, בכל פעם שהמשתמש בוחר באפשרות !Test מהתפריט.

שים לב:

העכבר הוא משאב משותף ולכן, הפיכת תפקוד הלחצנים שלו משפיעה על כל היישומים. התוכניות צריכות להימנע מהחלפת לחצני העכבר ככל האפשר. ובמקרה הצורך, להחזירם למצב המקורי.

הקשרי התקנים

בתוכנית מהסעיף הקודם, הושג הֶקשר הֶתקן לפני שהתוכנית שלחה פלט לחלון, והוא שוחרר לאחר שהפלט הופק. כעת נברר מהו **הֶקשר הֶתקן** (Device Context). למעשה זהו נתיב פלט העובר מהיישום, דרך הדרייבר המתאים להתקן, אל שטח הלקוח של החלון. הקשר ההתקן גם מגדיר בצורה מלאה את מצב הדרייבר של ההתקן.

לפני שהיישום שלך יוכל לשלוח מידע כפלט לשטח הלקוח של החלון, עליו להשיג הקשר התקן. עד אז, אין למעשה קשר בין התוכנית שלך לבין החלון, בכל הנוגע לפלט. לכן, חובה להשיג הקשר של התקן לפני משלוח הפלט לחלון. הפונקציה ()TextOut ופונקציות פלט אחרות דורשות ידית להקשר של התקן, לכן זהו כלל הנאכף מעצמו.



תרשים 3.1: כותרת חלון לדוגמה שהופק על ידי התוכנית WM_CHAR

3.10 עיבוד הודעה של WM_PAINT

לפני שתמשיך בקריאה, הרץ את התוכנית **Show_Keys** מהסעיף הקודם פעם נוספת, והקלד תווים אחדים. כעת, הקטן והגדל מחדש את החלון. כפי שיתברר לך, התו האחרון שהקלדת לא יוצג כשהחלון ישוחזר לגודלו הרגיל. זאת ועוד, אם החלון הוסתר על ידי חלון אחר ולאחר מכן נחשף מחדש, התו האחרון אינו מוצג עוד. הסיבה לכך פשוטה: ככלל, חלונות אינה רושמת לעצמה את תכולתו של חלון נתון. האחריות לשמירת תכולת החלון מוטלת על התוכנית שלך. כדי לסייע לתוכנית במשימה זו, כל פעם שמתעורר צורך להציג מחדש את תוכנו של החלון, תישלח לתוכנית שלך הודעת WM_PAINT (הודעה זו תישלח גם כאשר החלון מוצג לראשונה). כל פעם שהתוכנית מקבלת הודעה כזו, עליה להציג מחדש את תכולת החלון. בסעיף זה תוסיף לתוכנית משפט case, שמתפקידו לעבד את הודעת המאקרו WM_PAINT.

✍ **הערה:** מסיבות טכניות שונות, תכולתו של חלון מוצגת מחדש באופן אוטומטי לאחר הזזה של החלון. דבר זה לא יתרחש אם החלון הוסתר על ידי חלון אחר, או שחלקו היה מחוץ לתצוגה, והוצג מחדש.

לפני שנסביר כיצד להגיב על הודעת WM_PAINT, רצוי להסביר מדוע חלונות אינה כותבת מחדש את החלון באופן אוטומטי. התשובה לכך קצרה ועניינית: במקרים רבים, לתוכנית שלך (היודעת בדיוק מהי תכולתו של החלון), קל יותר מאשר לחלונות לכתוב מחדש את תכולת החלון. למרות שיתרונותיה של גישה זו הם נושא לוויכוח מתמשך בין מתכנתים, רצוי שפשוט תקבל זאת, כי אין זה סביר שהדבר ישתנה בקרוב.

הצעד הראשון בעיבוד הודעת WM_PAINT הוא הוספת משפט case למשפט switch בתוך פונקציית החלון, הנה כך:

```
case WM_PAINT: /* process a repaint request */
  hdc = BeginPaint(hwnd, &paintstruct); /* get DC */
  TextOut(hdc, 1, 1, str, strlen(str)); /* output string */
  EndPaint(hwnd, &paintstruct); /* release DC */
  break;
```

הבה נתבונן במשפט זה ביסודיות. ראשית, שים לב שהקשר ההתקן מושג באמצעות קריאה לפונקציה ()BeginPaint, ולא לפונקציה ()GetDC. מטעמים שונים, כאשר עתה עוסק בעיבוד של הודעת WM_PAINT, עליך להשיג את הקשר ההתקן באמצעות הפוקנציה ()BeginPaint, שהגדרתה היא:

```
HDC BeginPaint(HWND hwnd, LPPAINTSTRUCT lpPS);
```

הפרמטר השני הוא מצביע אל מבנה מסוג PAINTSTRUCT, שהגדרתו היא:

```
typedef struct tagPAINTSTRUCT {
  HDC hdc; /* handle to device context */
  BOOL fErase; /* true if background must be erased */
  RECT rcPaint; /* coordinates of region to redraw */
  BOOL fRestore;  /* reserved */
  BOOL fIncUpdate; /* reserved */
  BYTE rgbReserved[32]; /* reserved */
} PAINTSTRUCT;
```

הסוג RECT הוא מבנה המגדיר את הקואורדינטות לפינה השמאלית-העליונה והימנית-התחתונה של אזור מלבני. מבנה זה נראה כך:

```
typedef tagRECT {
  LONG left, top; /* upper left */
  LONG right, bottom; /* lower right */
} RECT;
```

במבנה PAINTSTRUCT, הרכיב rcPaint מכיל את קואורדינטות אזור החלון שיש לצובעו מחדש. לעת-עתה אינך צריך להשתמש בתכולתו של מבנה זה; תוכל להניח שיש להציג מחדש את כל החלון.

נציג תוכנית שלמה, המסוגלת לעבד הודעות מסוג WM_PAINT. התוכנית נמצאת בתיקיה Chap03**WM_Paint_chr**.

```
#include <windows.h>
#include "WM_Paint_chr.h"
#include <stdio.h>
#if defined (win32)
   #define IS_WIN32 TRUE
#else
   #define IS_WIN32 FALSE
#endif

HINSTANCE hInst;         // current instance
LPCTSTR lpszAppName = "WMPAINT";
LPCTSTR lpszTitle =   "WM_Paint";
BOOL RegisterWin95(CONST WNDCLASS* lpwc);
char str[80] = ""; /* holds output string */

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
 {
   MSG msg;
   HWND hWnd;
   WNDCLASS wc;

   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = (WNDPROC)WndProc;
   wc.cbClsExtra     = 0;
   wc.cbWndExtra     = 0;
   wc.hInstance      = 0;
   wc.hIcon          = LoadIcon(hInstance, lpszAppName);
   wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName   = lpszAppName;
   wc.lpszClassName  = lpszAppName;

   if(!RegisterWin95(&wc))
      return false;
   hInst = hInstance;
   hWnd = CreateWindow (lpszAppName,
                        lpszTitle,
```

```
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                       );
   if(!hWnd)
      return false;
   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
   while(GetMessage(&msg, NULL, 0,0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
   return(msg.wParam);
 }

BOOL RegisterWin95(CONST WNDCLASS* lpwc)
 {
   WNDCLASSEX wcex;

   wcex.style           = lpwc->style;
   wcex.lpfnWndProc     = lpwc->lpfnWndProc;
   wcex.cbClsExtra      = lpwc->cbClsExtra;
   wcex.cbWndExtra      = lpwc->cbWndExtra;
   wcex.hInstance       = lpwc->hInstance;
   wcex.hIcon           = lpwc->hIcon;
   wcex.hCursor         = lpwc->hCursor;
   wcex.hbrBackground   = lpwc->hbrBackground;
   wcex.lpszMenuName    = lpwc->lpszMenuName;
   wcex.lpszClassName   = lpwc->lpszClassName;
   wcex.cbSize          = sizeof(WNDCLASSEX);
   wcex.hIconSm         = LoadIcon(wcex.hInstance, "SMALL");
   return RegisterClassEx(&wcex);
 }
```

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam)
 {
   HDC hdc;
   PAINTSTRUCT paintstruct;
   switch(uMsg)
    {
    case WM_CHAR: /* process keystroke */
      hdc = GetDC(hWnd); /* get device context */
      TextOut(hdc, 1, 1, "  ", 2); /* erase old character */
      sprintf(str, "%c", (char) wParam);
                                   /* stringize character */
      TextOut(hdc, 1, 1, str, strlen(str)); /* output char */
      ReleaseDC(hWnd, hdc); /* release device context */
      break;
    case WM_PAINT: /* process a repaint request */
      hdc = BeginPaint(hWnd, &paintstruct); /* get DC */
      TextOut(hdc, 1, 1, str, strlen(str)); /* output string*/
      EndPaint(hWnd, &paintstruct); /* release DC */
      break;

      case WM_COMMAND:
         switch(LOWORD(wParam))
          {
            case IDM_TEST :
               break;
            case IDM_EXIT :
               DestroyWindow(hWnd);
               break;
          }
         break;
      case WM_DESTROY :
         PostQuitMessage(0);
         break;
      default:
         return (DefWindowProc(hWnd, uMsg, wParam, lParam));
    }
   return(0L);
 }
```

לפני שתמשיך, הקלד, הדר והרץ תוכנית זו. נסה להקליד כמה תווים ולאחר מכן להקטין את החלון ולשחזרו לגודלו הרגיל. כפי שיתברר לך, בכל פעם שהחלון יוצג מחדש, התו האחרון שהקלדת יצויר מחדש באופן אוטומטי. שים לב שהמערך הגלובלי str מאותחל ל-Sample Output, ושמחרוזת זאת מוצגת בעת שהתוכנית מתחילה לרוץ. הסיבה לכך היא, שהודעת WM_PAINT מופקת באופן אוטומטי כל פעם שחלון נוצר.

הטיפול בהודעות של WM_PAINT בתוכנית השלד פשוט למדי, ראוי להדגיש שבמרבית הגרסאות הממשיות, הדברים יהיו מורכבים הרבה יותר, כי רוב החלונות מכילים הרבה יותר פלט.

האחריות לשחזור התכולה של החלון במקרה של שינוי גודל, או הסתרה מוטלת על התוכנית שלך. מחובתך לספק לכל תוכנית מנגנון כלשהו שבאמצעותו היא תוכל לבצע זאת. בתוכניות אמיתיות, הדבר נעשה באחת משלוש דרכים:

1. התוכנית יכולה להפיק מחדש את הפלט באמצעות חישובים מתאימים. דרך זו מתאימה כשאין צורך בפעולה כלשהי מצד המשתמש.
2. במקרים מסוימים אפשר לרשום ולשמור את רצף האירועים, ולהריץ אותם מחדש כל פעם שצריך לצייר מחדש את החלון.
3. התוכנית יכולה לשמור מסך וירטואלי, שתכולתו מועתקת אל החלון בכל פעם שצריך לציירו מחדש.

השיטה השלישית היא הכללית ביותר מבין השלוש (יישומה של גישה זו מתואר בהמשך הספר). איזו גישה היא הטובה ביותר? זה תלוי לחלוטין ביישום. ברוב הדוגמאות בספר זה איננו טורחים לצייר מחדש את החלון, מכיון שהדבר כרוך בתוספת ניכרת של קוד, שרק תטשטש את הנקודה שאנו מנסים להבליט בכל דוגמה. כדי ליצור יישומי חלונות 9x תקניים, תצטרך לדאוג לשחזור החלונות בתוכניות שלך.

3.11 כיצד להפיק הודעת WM_PAINT

תוכל לגרום לתוכנית שלך להפיק הודעת WM_PAINT. ייתכן שתשאל את עצמך מדוע שיהיה צורך בכך, זאת מכיון שהתוכנית יכולה לצבוע מחדש את החלון שלה בכל עת שתרצה. אבל זוהי הנחה שגויה. זכור, עדכון של חלון הוא תהליך יקר במונחי זמן. חלונות היא מערכת לריבוי משימות וייתכן שהיא מריצה במקביל תוכניות נוספות שגם להן דרוש זמן CPU, ולכן רצוי שהתוכנית שלך רק תגיד למערכת ההפעלה שהיא מעוניינת להפיק פלט, אך תניח לה להחליט מתי לבצע את הדבר בפועל. דבר זה מאפשר לחלונות לנהל את המערכת בצורה יעילה יותר ולהקצות זמן CPU לכל המשימות הרצות בה. על פי גישה זו, התוכנית שלך תעכב כל פלט עד שתקבל הודעת WM_PAINT ממערכת ההפעלה.

בדוגמאות הקודמות התקבלה הודעה כזאת רק כשהחלון נחשף מחדש לאחר שהיה מוסתר, או כאשר שוחזר ממצב מוקטן לגודל רגיל. אולם, אם מעובב כל הפלט עד לקבלת הודעת WM_PAINT ואז מושג פלט/קלט אינטראקטיבי, חייבת להיות דרך

כלשהי להודיע לחלונות שהיא צריכה לשלוח הודעת WM_PAINT לחלון שלך בכל פעם שיש צורך בהפקת פלט. כצפוי, תכונה זו גלומה בחלונות 9x. וכך, בכל פעם שיש לתוכנית שלך מידע שהיא רוצה להוציא כפלט, היא רק מבקשת ממערכת ההפעלה לשלוח לה הודעת WM_PAINT כאשר תהיה פנויה לעשות זאת. כדי לגרום למערכת ההפעלה לשלוח הודעת WM_PAINT, תפעיל התוכנית שלך פונקציית API בשם InvalidateRect(). הגדרתה מובאת כאן:

```
BOOL InvalidateRect(HWND hwnd, CONST RECT *lpRect, BOOL bErase);
```

כאן, הפרמטר hwnd הוא הידית לחלון שאליו אתה רוצה לשלוח את הודעת WM_PAINT. הפרמטר lpRect מצביע על המבנה RECT, המכיל את קואורדינטות שטח החלון, אשר יש לציירו מחדש. ערך NULL בפרמטר זה מציין את החלון כולו. אם הפרמטר bErase מכיל ערך true, יימחק הרקע של החלון. אם הוא מכיל ערך אפס, לא יחול שום שינוי ברקע. פונקציה זו מחזירה ערך לא-אפס אם סיימה בהצלחה, וערך אפס אם לא סיימה בהצלחה (בדרך כלל הפונקציה מסיימת בהצלחה).

כאשר הפונקציה InvalidateRect() מופעלת, היא מודיעה לחלונות שהחלון אינו תקף עוד ויש לציירו מחדש. דבר זה, גורם למערכת ההפעלה לשלוח הודעת WM_PAINT לפונקציית החלון של התוכנית.

לפניך גירסה חדשה של שלד היישום הקודם, המבצעת כל פלט על ידי הפקת הודעת WM_PAINT. יתר הקוד לתגובה על הודעות מכין את המידע המיועד להצגה, ולאחר מכן מפעיל את הפונקציה InvalidateRect(). התוכנית נמצאת בתיקיה Chap03**WM_Paint**.

```
#include <windows.h>
#include "WM_Paint.h"
#include <stdio.h>
#if defined (win32)
   #define IS_WIN32 TRUE
#else
   #define IS_WIN32 FALSE
#endif

HINSTANCE hInst;         // current instance
LPCTSTR lpszAppName = "WMPAINT";
LPCTSTR lpszTitle =   "WM_Paint";
BOOL RegisterWin95(CONST WNDCLASS* lpwc);
char str[80] = ""; /* holds output string */

int X = 1, Y = 1; /* screen location */

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE
                    hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
 {
```

```
  MSG msg;
  HWND hWnd;
  WNDCLASS wc;

  wc.style          = CS_HREDRAW | CS_VREDRAW;
  wc.lpfnWndProc    = (WNDPROC)WndProc;
  wc.cbClsExtra     = 0;
  wc.cbWndExtra     = 0;
  wc.hInstance      = 0;
  wc.hIcon          = LoadIcon(hInstance, lpszAppName);
  wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
  wc.lpszMenuName   = lpszAppName;
  wc.lpszClassName  = lpszAppName;

  if(!RegisterWin95(&wc))
     return false;
  hInst = hInstance;
  hWnd = CreateWindow (lpszAppName,
                       lpszTitle,
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, 0,
                       CW_USEDEFAULT, 0,
                       NULL,
                       NULL,
                       hInstance,
                       NULL
                      );
  if(!hWnd)
     return false;
  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);
  while(GetMessage(&msg, NULL, 0,0))
   {
     TranslateMessage(&msg);
     DispatchMessage(&msg);
   }
  return(msg.wParam);
}
```

```
BOOL RegisterWin95(CONST WNDCLASS* lpwc)
 {
   WNDCLASSEX wcex;

   wcex.style           = lpwc->style;
   wcex.lpfnWndProc     = lpwc->lpfnWndProc;
   wcex.cbClsExtra      = lpwc->cbClsExtra;
   wcex.cbWndExtra      = lpwc->cbWndExtra;
   wcex.hInstance       = lpwc->hInstance;
   wcex.hIcon           = lpwc->hIcon;
   wcex.hCursor         = lpwc->hCursor;
   wcex.hbrBackground   = lpwc->hbrBackground;
   wcex.lpszMenuName    = lpwc->lpszMenuName;
   wcex.lpszClassName   = lpwc->lpszClassName;
   wcex.cbSize          = sizeof(WNDCLASSEX);
   wcex.hIconSm         = LoadIcon(wcex.hInstance, "SMALL");
   return RegisterClassEx(&wcex);
 }

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
LPARAM lParam)
 {
   HDC hdc;
   PAINTSTRUCT paintstruct;
   switch(uMsg)
    {
         case WM_CHAR: /* process keystroke */
        X = Y = 1; /* display chars in upper left corner */
        sprintf(str, "%c", (char) wParam);
                 /* stringize character */
        InvalidateRect(hWnd, NULL, 1); /* paint the screen */
        break;
      case WM_PAINT: /* process a repaint request */
        hdc = BeginPaint(hWnd, &paintstruct); /* get DC */
        TextOut(hdc, X, Y, str, strlen(str)); /* output string*/
        EndPaint(hWnd, &paintstruct); /* release DC */
        break;
          case WM_RBUTTONDOWN: /* process right button */
               strcpy(str, "Right button is down.");
               X = LOWORD(lParam); /* set X,Y to current */
```

```
            Y = HIWORD(lParam); /* mouse location */
            InvalidateRect(hWnd, NULL, 1);
                                    /* paint the screen */
            break;
    case WM_LBUTTONDOWN: /* process left button */
            strcpy(str, "Left button is down.");
            X = LOWORD(lParam); /* set X,Y to current */
            Y = HIWORD(lParam); /* mouse location */
            InvalidateRect(hWnd, NULL, 1);
                                    /* paint the screen */
            break;
    case WM_COMMAND:
        switch(LOWORD(wParam))
         {
          case IDM_TEST :
            break;
          case IDM_EXIT :
            DestroyWindow(hWnd);
            break;
         }
        break;
    case WM_DESTROY :
        PostQuitMessage(0);
        break;
    default:
        return (DefWindowProc(hWnd, uMsg, wParam, lParam));
  }
  return(0L);
}
```

שים לב שהתוספו לתוכנית שני משתנים גלובליים חדשים, X ו-Y, המחזיקים את המיקום שבו יוצג הטקסט כאשר תתקבל הודעת WM_PAINT.

כפי שאתה רואה, על ידי ניתוב כל הפלט דרך WM_PAINT, למעשה צמצמנו את גודל התוכנית, ובמובנים מסוימים פישטנו אותה. כפי שכבר הזכרנו, התוכנית גם מאפשרת באמצעות נוהל זה לחלונות 9x להחליט על המועד המתאים ביותר לעדכון החלון.

✍ **הערה:** יישומי חלונות רבים מנתבים את כל (או רוב) הפלט דרך WM_PAINT, מהסיבות שפורטו לעיל. טכנית, אין שום פסול בכך שהתוכניות הישנות מפיקות טקסט בעת שהן מגיבות על הודעה, גם אם גישה זו אינה הגישה הטובה ביותר לכל מצב.

3.12 הפקת הודעות של קוצב זמן

ההודעה האחרונה שתידון כאן היא הודעת WM_TIMER. באמצעות חלונות, תוכל ליצור **קוצב זמן** (Timer), שייצור **פסיקות** (Interrupts) בתוכנית שלך במרווחי זמן קבועים מראש. בכל פעם שקוצב הזמן מופעל, הוא שולח הודעת WM_TIMER לפונקציית החלון שלך. השימוש בקוצב זמן הוא שיטה טובה "לעורר" את התוכנית בפרקי זמן קצובים. דבר זה שימושי במיוחד כשהתוכנית שלך רצה בתור משימת רקע.

כדי להפעיל קוצב זמן, הפעל פונקציית API בשם ()SetTimer, שהגדרתה היא:

```
UINT SetTimer(HWND hwnd, UINT nID, UINT wLength, TIMERPROC
lpTFunc);
```

הפרמטר hwnd הוא הידית לחלון שמשתמש בקוצב הזמן. הפרמטר nID מציין את הערך שיקשר עם קוצב הזמן הזה (מותר להפעיל בו-זמנית יותר מקוצב זמן אחד). הערך השמור ב-wLength מציין את פרק הזמן באלפיות השנייה, כלומר, כמה זמן יעבור בין פסיקה לפסיקה. lpTFunc הוא **מצביע**, המצביע על פונקציית קוצב הזמן שתופעל כל פעם שקוצב הזמן "מצלצל". פונקציה זו חייבת להיות **פונקציית משוב** (Callback) המחזירה ערך VOID CALLBACK, ומקבלת אותו סוג של פרמטרים כמו פונקציית החלון. אך, אם הערך של lpTFunc הוא NULL, כפי שקורה פעמים רבות, תשמש לצורך זה פונקציית החלון של התוכנית. במקרה זה, כל פעם שקוצב הזמן "יצלצל", תישלח הודעת WM_TIMER לתור ההודעות של התוכנית שלך, ופונקציית החלון של התוכנית תעבד אותה כמו כל הודעה אחרת. זוהי הגישה שננקוט בדוגמה הבאה. הפונקציה מחזירה nID אם סיימה בהצלחה. אם לא ניתן להקצות קוצב זמן, מוחזר ערך אפס.

לאחר שכיוונת קוצב זמן, הוא ימשיך להפסיק את התוכנית שלך בפרקי הזמן הקצובים עד שתסיים את היישום, או שהתוכנית שלך תקרא לפונקציה ()KillTimer, שהגדרתה היא:

```
BOOL KillTimer(HWND hwnd, UINT nID);
```

הפרמטר hwnd הוא החלון המכיל את קוצב הזמן, ואילו nID הוא הערך המזהה את קוצב הזמן המסוים הזה.

כל פעם שמופקת הודעת WM_TIMER, נשמר בפרמטר wParam ערך ID של קוצב הזמן, ואילו lParam מכיל את המען של פונקציית המשוב של קוצב הזמן (אם הוגדרה פונקציה כזאת). בדוגמה הבאה, יכיל הפרמטר lParam את הערך NULL.

כדי להדגים את השימוש בקוצב זמן, משתמשת התוכנית **WM_Timer** שלפניך (בתיקיה Chap03) בקוצב זמן ליצירת שעון. לשם כך היא משתמשת בפונקציות השעה והתאריך התקניות של ++C/C, כדי לקלוט ולהציג את זמן המערכת ותאריך המערכת הנוכחיים. כל פעם שקוצב הזמן "מצלצל", במקרה זה בערך פעם בשנייה, מתעדכנת השעה. באופן זה, השעה המוצגת על המסך מדויקת בגבולות של שנייה אחת.

```
#include <windows.h>
#include "WM_Timer.h"
#include <stdio.h>
#include <time.h>

#if defined (win32)
   #define IS_WIN32 TRUE
#else
   #define IS_WIN32 FALSE
#endif

HINSTANCE hInst;        // current instance
LPCTSTR lpszAppName = "WMPAINT";
LPCTSTR lpszTitle =   "WM_Paint";
BOOL RegisterWin95(CONST WNDCLASS* lpwc);
char str[80] = ""; /* holds output string */

int X = 1, Y = 1; /* screen location */

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
 {
   MSG msg;
   HWND hWnd;
   WNDCLASS wc;

   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = (WNDPROC)WndProc;
   wc.cbClsExtra     = 0;
   wc.cbWndExtra     = 0;
   wc.hInstance      = 0;
   wc.hIcon          = LoadIcon(hInstance, lpszAppName);
   wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName   = lpszAppName;
   wc.lpszClassName  = lpszAppName;
```

```
    if(!RegisterWin95(&wc))
        return false;
    hInst = hInstance;
    hWnd = CreateWindow (lpszAppName,
                         lpszTitle,
                         WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT, 0,
                         CW_USEDEFAULT, 0,
                         NULL,
                         NULL,
                         hInstance,
                         NULL
                        );
    if(!hWnd)
        return false;
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    /* start a timer -- interrupt once per second */
    SetTimer(hWnd, 1, 1000, NULL);

    while(GetMessage(&msg, NULL, 0,0))
     {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
     }

    KillTimer(hWnd, 1); /* stop the timer */
    return(msg.wParam);
 }

BOOL RegisterWin95(CONST WNDCLASS* lpwc)
 {
    WNDCLASSEX wcex;

    wcex.style           = lpwc->style;
    wcex.lpfnWndProc     = lpwc->lpfnWndProc;
    wcex.cbClsExtra      = lpwc->cbClsExtra;
    wcex.cbWndExtra      = lpwc->cbWndExtra;
    wcex.hInstance       = lpwc->hInstance;
```

```
    wcex.hIcon            = lpwc->hIcon;
    wcex.hCursor          = lpwc->hCursor;
    wcex.hbrBackground    = lpwc->hbrBackground;
    wcex.lpszMenuName     = lpwc->lpszMenuName;
    wcex.lpszClassName    = lpwc->lpszClassName;
    wcex.cbSize           = sizeof(WNDCLASSEX);
    wcex.hIconSm          = LoadIcon(wcex.hInstance, "SMALL");
    return RegisterClassEx(&wcex);
  }

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam)
  {
    HDC hdc;
    PAINTSTRUCT paintstruct;
    time_t t;
    struct tm *newtime;

    switch(uMsg)
     {
          case WM_PAINT: /* process a repaint request */
               hdc = BeginPaint(hWnd, &paintstruct); /* get DC*/
               TextOut(hdc, X, Y, str, strlen(str));
                         /* output string */
               EndPaint(hWnd, &paintstruct); /* release DC */
               break;
       case WM_TIMER: /* timer went off */
               /* get the new time */
               t = time(NULL);
               newtime = localtime(&t);

               /* display the new time */
               strcpy(str, asctime(newtime));
               str[strlen(str)-1] = '\0'; /* remove /r/n */
               InvalidateRect(hWnd, NULL, 0); /* update screen */
               break;
```

```
        case WM_DESTROY :
                PostQuitMessage(0);
        break;
     default:
        return (DefWindowProc(hWnd, uMsg, wParam, lParam));
   }
  return(0L);
}
```

לאחר שלמדת כיצד תוכנית של חלונות 9x מעבדת הודעות, תוכל להמשיך וליצור תיבות הודעה ותפריטים, נושאים שיידונו בפרק 4.

פרק 4

תיבות הודעה ותפריטים

עכשיו, כשאתה כבר יודע כיצד לבנות שלד בסיסי לתוכנית חלונות שתדע לקבל ולעבד הודעות, הגיע הזמן להתחיל בבדיקת המרכיבים של ממשק המשתמש של חלונות. אם אתה לומד תכנות חלונאי לראשונה, חשוב שתבין שהיישומים שלך יתקשרו עם המשתמש בדרך כלל באמצעות אחד, או יותר ממרכיבי הממשק המוגדרים מראש. חלונות 9x תומכת בכמה סוגים של מרכיבי ממשק. פרק זה דן בשניים מהם: **תיבות הודעה ותפריטים**. אלה הם מרכיבי הממשק הבסיסיים ביותר של חלונות, שיופיעו כמעט בכל תוכנית שתכתוב. כפי שיתברר לך, הסגנון הבסיסי של תיבת ההודעה ושל התפריט מוגדר מראש. מה שנותר לך לעשות הוא לספק את המידע המתייחס ליישום שלך. בפרק זה תכיר את המושג **משאב** (Resource), מרכיב חיוני ברוב יישומי חלונות.

4.1 הפונקציה MessageBox

החלון הפשוט ביותר בממשק הוא תיבת ההודעה. ***תיבת הודעה*** (Message Box) מציגה הודעה למשתמש, ודורשת תגובה מהמשתמש לפני שהתוכנית תוכל להמשיך בפעולתה. בסעיפים שנראה בהמשך נלמד ליצור תיבות דו-שיח מסוגים שונים. עם זאת, עבור תיבות דו-שיח פשוטות המקבלות קלט מהמשתמש, התוכניות יכולות להשתמש בפונקציה MessageBox. פונקציה זו יוצרת, מציגה ומפעילה ***תיבת הודעה*** (Message Box). תיבת הודעה מכילה הודעה וכותרת שמוגדרים על ידי היישום, בנוסף לצירופים מובנים של סמלים ולחצנים. תיבת ההודעה אינה יכולה להציג חלונות בעצמה. התוכניות שתכתוב תשתמשנה בצורה הכללית של הפונקציה MessageBox כדי להציג תיבות הודעה, כפי שתראה להלן:

```
int MessageBox(
  HWND  hWnd,          // handle of owner window
  LPCTSTR  lpText,    // address of text in message box
  LPCTSTR  lpCaption, // address of title of message // box
  UINT uType           // style of message box
  );
```

הפרמטרים שהפונקציה MessageBox מקבלת מוסברים ברשימה שבטבלה 4.1.

נוסף לטקסט שתיבת ההודעה מציגה (כפי שנקבע על ידי הפרמטר lpText) והכותרת עבור תיבת ההודעה (שנקבעת על ידי הפרמטר lpCaption), התוכניות תשתמשנה פעמים רבות בפרמטר uType כדי לשנות את הלחצנים והסמלים שרוצים להציג בתיבת ההודעה. טבלה 4.2 מפרטת את הערכים האפשריים שמקבל הפרמטר uType לשליטה במספר הלחצנים וסוגיהם שמוצגים בתיבת ההודעה.

טבלה 4.1: הפרמטרים שהפונקציה **MessageBox** מקבלת.

דגל	תיאור
hWnd	מזהה את החלון לו תהיה שייכת תיבת ההודעה שרוצים ליצור. אם פרמטר זה הוא NULL, פירוש הדבר שאין לתיבת ההודעה חלון לו היא שייכת.
lpText	מצביע על מחרוזת המסתיימת ב-NULL שמכילה את ההודעה שתיבת ההודעה צריכה להציג.
lpCaption	מצביע על מחרוזת המסתיימת ב-NULL שמשמשת את כותרת תיבת ההודעה. אם פרמטר זה הוא NULL, התוכנית משתמשת בכותרת ברירת המחדל שגיאה (Error).
uType	מגדיר קבוצה של דגלי סיביות (Bit Flags) הקובעים את תכולתה והתנהגותה של תיבת ההודעה. פרמטר זה יכול להיות שילוב של דגלים מקבוצת הדגלים. טבלה 4.2 מציגה את רשימת הדגלים שהתוכנית יכולה להשתמש בהם, כדי לציין את הלחצנים המוכלים בתיבת ההודעה.

טבלה 4.2: הערכים האפשריים שמקבל הפרמטר **uType** לשליטה במספר הלחצנים.

דגל	פירוש
MB_ABORTRETRYIGNORE	תיבת ההודעה מכילה שלושה לחצנים: **בטל** (Abort), **נסה שנית** (Retry), ו**התעלם** (Ignore).
MB_OK	תיבת ההודעה מכילה לחצן אחד: **אישור** (OK). הדגל MB_OK הוא ברירת המחדל של הפרמטר uType.
MB_OKCANCEL	תיבת ההודעה מכילה שני לחצנים: **אישור** (OK) ו**ביטול** (Cancel).
MB_RETRYCANCEL	תיבת ההודעה מכילה שני לחצנים: **נסה שנית** (Retry) ו**ביטול** (Cancel).
MB_YESNO	תיבת ההודעה מכילה שני לחצנים: **כן** (Yes), ו**לא** (No).
MB_YESNOCANCEL	תיבת ההודעה מכילה שלושה לחצנים: **כן** (Yes), ו**לא** (No) ו**ביטול** (Cancel).

בנוסף לשליטה בכותרת ובמספר הלחצנים שמוצגים עם תיבת ההודעה, באפשרות התוכנית גם לקבוע אם יוצג סמל עם תיבת ההודעה, או לא. ברירת המחדל היא MB_NOICON (כלומר, לא). טבלה 4.3 מציגה רשימה של הערכים האפשריים שמקבל הפרמטר uType לשליטה בהופעת הסמל עם תיבת ההודעה.

טבלה 4.3: הערכים האפשריים שמקבל הפרמטר **uType** לשליטה בסמל תיבת ההודעה.

דגל	פירוש
MB_ICONEXCLAMATION	מציג בתיבת ההודעה סמל סימן קריאה.
MB_ICONWARNING	מציג בתיבת ההודעה סמל סימן קריאה.
MB_ICONINFORMATION	מציג בתיבת ההודעה סמל שמורכב מהאות i בתוך מעגל.
MB_ICONASTERISK	מציג בתיבת ההודעה סמל שמורכב מהאות i בתוך מעגל.
MB_ICONQUESTION	מציג בתיבת ההודעה סמל סימן שאלה.
MB_ICONSTOP	מציג בתיבת ההודעה סמל עצור.
MB_ICONERROR	מציג בתיבת ההודעה סמל עצור.
MB_ICONHAND	מציג בתיבת ההודעה סמל עצור.
MB_NOICON	לא מוצג סמל כלשהו בתיבת ההודעה.

הפונקציה MessageBox תומכת במספר רב של קבועים נוספים עבור הפרמטר uType. אך הקבועים שנמצאים בטבלה 4.2 ובטבלה 4.3 הם השימושיים ביותר. כדי לראות את רשימת כל הקבועים בדוק את העזרה המקוונת של המהדר. הפונקציה ()MessageBox מחזירה את תגובת המשתמש לתיבה. הערכים החוזרים האפשריים הם:

טבלה 4.4

הלחצן שנבחר	הערך החוזר
Abort	IDABORT
Retry	IDRETRY
Ignore	IDIGNORE
Cancel	IDCANCEL
No	IDNo
Yes	IDYes
OK	IDOK

כדי להציג תיבת הודעה, הפעל את הפונקציה ()MessageBox. חלונות תציג את תיבת ההודעה בהזדמנות הראשונה שתוכל. אינך צריך להשיג הקשר התקן, או להפיק הודעת WM_PAINT. הפונקציה ()MessageBox מטפלת בכל הפרטים האלה במקומך (כיון שכה פשוט להשתמש בתיבות הודעה, הן משמשות כלי נפלא לניפוי תוכנה, כאשר דרוש לך אמצעי פשוט להוציא פלט אל המסך).

התקליטור שמצורף לספר זה מכיל את התוכנית **Show_Mess** (בתיקיה chap04). התוכנית **Show_Mess** אומרת "!Hello" למשתמש וממתינה להקשה על מקש כלשהו.

4.2 הפונקציה MessageBeep

בסעיף הקודם למדת להשתמש בפונקציה MessageBox כדי ליצור תיבת דו-שיח פשוטה. התוכנית **Show_Mess** קוראת לפונקציה MessageBeep כשהיא צריכה ליצור תיבת ההודעה. הפונקציה MessageBeep משמיעה נגינה בפורמט WAVE (Waveform). פריט בקטע [Sound] של מערכת הרישום מגדירה את תבנית פורמט גל הקול (Waveform Sound) עבור כל סוג של קול. התוכנית תשתמש בפונקציה MessageBeep לפי ההגדרה שלהלן:

```
BOOL MessageBeep(UINT uType);
```

הפרמטר uType מגדיר את סוג הקול, כמו שמזוהה על ידי הכניסה בחלק [Sound] של רישום המערכת. פרמטר הקול uType יכול לקבל את אחד הערכים שבטבלה 4.5.

בטבלה 4.5: ערכים אפשריים של פרמטר הקול uType.

ערך	קול
0xFFFFFFFF	צליל סטנדרטי המושמע על ידי רמקול המחשב.
MB_ICONASTERISK	SystemAsterisk (כוכבית).
MB_ICONEXCLAMATION	SystemExclamation (סימן קריאה).
MB_ICONHAND	SystemHand (יד).
MB_ICONQUESTION	SystemQuestion (שאלה).
MB_OK	SystemDefault (ברירת מחדל).

התוכניות צריכות להשתמש בפונקציה MessageBeep כדי להשמיע קול פשוט, שמשתמשים בו בדרך כלל להתריע למשתמש על מאורע כלשהו.

לפניך דוגמה פשוטה המציגה תיבת הודעה בכל פעם שאתה לוחץ על אחד מלחצני העכבר (הבאנו פה רק את לולאת ההודעות של התוכנית **MessageBox**. התוכנית בשלמותה נמצאת בתיקיה : Chap04\MessageBox):

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam )
{
   int response;
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        MessageBeep( MB_ICONEXCLAMATION );
                        if ( MessageBox( hWnd, "Hello!", NULL,
                          MB_YESNO | MB_ICONEXCLAMATION) == IDYES)
                        {
                           // YES was pressed..
                        }
                        break;

                 case IDM_ABOUT :
                        DialogBox( hInst, "AboutBox", hWnd,
                                 (DLGPROC)About );
                        break;

                 case IDM_EXIT :
                        DestroyWindow( hWnd );
                        break;
              }
              break;
      case WM_RBUTTONDOWN: /* process right button */
                             response = MessageBox(hWnd,
                                  "Press One:", "Right Button",
                                   MB_ABORTRETRYIGNORE);
```

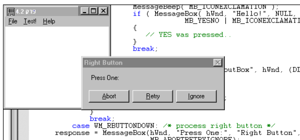
```
                                switch(response) {
                                    case IDABORT:
                                       MessageBox(hWnd, "",
                                                  "Abort", MB_OK);
                                       break;
                                    case IDRETRY:
                                       MessageBox(hWnd, "",
                                                  "Retry", MB_OK);
                                       break;
                                    case IDIGNORE:
                                       MessageBox(hWnd, "",
                                                  "Ignore", MB_OK);
                                       break;
                                }
                                break;
            case WM_LBUTTONDOWN: /* process left button */
                                      response = MessageBox(hWnd,
                                        "Continue?", "Left Button",
                                        MB_ICONHAND | MB_YESNO);
                                      switch(response) {
                                          case IDYES:
                                               MessageBox(hWnd,
                                                     "Press Button",
                                                     "Yes", MB_OK);
                                              break;
                                          case IDNO:
                                             MessageBox(hWnd,
                                                     "Press Button",
                                                     "No", MB_OK);
                                             break;
                                          }
                                          break;

      case WM_DESTROY :
              PostQuitMessage(0);
              break;

      default :
          return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
   }

   return( 0L );
}
```

בכל פעם שאתה לוחץ על לחצן, מופיעה על המסך תיבת הודעה. למשל, לחיצה על הלחצן הימני של העכבר תגרום להצגת תיבת הודעה על המסך, לדוגמה:



תרשים 4.1

כפי שאתה רואה, בתיבת ההודעה מופיעים הלחצנים Abort, Retry ו-Ignore. בהתאם לתגובה שלך, תוצג תיבת הודעה שנייה, שבה יצוין על איזה לחצן לחצת. לחיצה על הלחצן השמאלי של העכבר תגרום להופעת תיבת הודעה ובה סמל "עצור". תיבה זו מאפשרת לך להגיב ב-Yes או No.

לפני שתמשיך, ערוך ניסיונות עם תיבות הודעה, נסה לגרום להצגת סוגים שונים של תיבות הודעה על המסך שלך.

4.3 היכרות עם תפריטים

אמצעי השליטה הנפוץ ביותר בחלונות הוא התפריט. כמעט כל החלונות הראשיים מלווים בתפריט מסוג כלשהו. תפריטים הם פריט שכיח וחשוב ביישומים, ולכן יש במערכת ההפעלה תמיכה מובנית ומקיפה באמצעי זה. כפי שתראה, הוספת תפריט לחלון נתון כרוכה בשלבים הבאים:

1. הגדרת התצורה של התפריט בקובץ משאבים.
2. טעינת התפריט כל פעם שהתוכנית שלך יוצרת את החלון הראשי שלה.
3. עיבוד הוראות הנובעות מבחירת אחד הפריטים מתוך התפריט.

ככלל, תפריט הרמה העליונה מוצג לאורך חלקו העליון של החלון. תפריטי-משנה יופיעו בתצורה **נשלפת** (Popup). כל שנאמר כאן אינו חדש עבורך, כי כך נוהגים ברוב התוכניות.

לפני שנמשיך, חשוב להבין מהם משאבים וקבצי משאבים.

4.4 כיצד להשתמש במשאבים

מערכת ההפעלה חלונות מגדירה מספר סוגים נפוצים של עצמים כ**משאבים** (Resources). בעיקרון, משאבים הם עצמים המשמשים את התוכנית שלך, אך הוגדרו מחוצה לה. בין אלה ניתן למנות פריטים כגון תפריטים, סמלים, תיבות דו-שיח וגרפיקת מפת-סיביות. הואיל ותפריט נחשב למשאב, עליך להבין מהם משאבים בטרם תוכל להוסיף תפריט לתוכניתך.

משאב הוא עצם הנוצר בנפרד מהתוכנית שלך, אך מתוסף לקובץ EXE כאשר התוכנית שלך מקושרת. משאבים נשמרים בתוך **קבצי משאבים** (Resource Files), שהם קבצים בעלי סיומת RC. ככלל, שם הקובץ אמור להיות זהה לשם קובץ ההפעלה של התוכנית. לדוגמה, אם התוכנית שלך נקראת PROG.EXE, קובץ המשאבים שלה צריך להיקרא PROG.RC.

חלק מהמשאבים הם **קבצי טקסט** שאתה יוצר בעזרת עורך הטקסט הרגיל. משאבי טקסט מוגדרים בדרך כלל בתוך קובץ המשאבים. משאבים מסוגים אחרים, כגון סמלים, קל יותר להפיק בעזרת **עורך משאבים** (Resource Editor), אבל יחד עם זאת, חובה לאזכר אותם בקובץ המשאבים הקשור ביישום שלך. קבצי המשאבים המובאים לדוגמה בפרק זה הם קבצי טקסט רגילים, כיון שתפריטים הם משאבים המבוססים על טקסט.

קבצי משאבים אינם כתובים ב- C או ב- ++C, אלא בשפת משאבים מיוחדת, או **שפת תסריט** (script), ויש להדרם באמצעות **מהדר משאבים** (Resource Compiler). מהדר המשאבים הופך קבצי RC לקבצי RES, אותם ניתן **לקשר** (Link) עם התוכניות שלך.

כיצד להדר קבצי RC

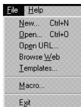
תוכניות אינן משתמשות במישרין בקבצי משאבים. כדי שיוכלו לשמש את התוכניות, יש להמיר את קבצי המשאבים למתכונת ניתנת לקישור. לאחר שיצרת קובץ RC, עליך להדר אותו ולהופכו לקובץ RES בעזרת מהדר המשאבים (המכונה לרוב RC.EXE, אבל אין זו חובה). כיצד בדיוק תהדר את קובץ המשאבים? דבר זה תלוי במהדר שלך. זאת ועוד, בחלק מסביבות הפיתוח המשולבות, שלב זה מתבצע באופן אוטומטי על ידי המערכת. בכל מקרה, הפלט של מהדר המשאבים יהיה קובץ בעל סיומת RES, וזהו הקובץ שאותו תוכל לקשר עם התוכנית כדי לסיים את בניית היישום לחלונות 9x.

✍ **הערה:** רצוי לעיין במדריך למשתמש הנלווה למהדר שלך. שם תמצא הוראות כיצד לכלול קבצי משאבים בתוכניות.

4.5 סוגי תפריטים (Menu Types)

שורת התפריט היא למעשה **מכולה** (Container) למשאבי תפריט. ככלל, מרכזים את התפריטים בקבוצות שמספקות למשתמש מספר אפשרויות.

תוכניות Windows משתמשות בשני סוגים עיקריים של תפריטים: **תפריטים עליונים** (Top-Level Menus), שמוצגים תמיד, ו**תפריטים נשלפים** או **מוקפצים** (Pop-Up Menus) שמוצגים בעת הצורך בלבד. התפריט העליון שנראה תמיד (נקרא גם התפריט "**הראשי**" של התוכנית) הוא למעשה קבוצת פקודות שנראות בחלון שורת התפריט כל הזמן, בהנחה שהתוכנית משתמשת בתפריט. עבור תוכניות מורכבות יותר, ייתכן שלא יהיה אפשרי או מעשי, להכניס את כל פריטי התפריט לשורה אחת. במקרים אלה התוכניות יכולה להשתמש ב**תפריטי משנה** (Sub-Menus) שמוצגים רק כאשר פונים לפריט האב שלהם, או שמוצגים **תפריטים נפתחים**, או **נשלפים למטה** (Pull-Down Menus), או **תפריטים הנזרקים למטה** (Drop-Down Menus). כשבוחרים אפשרות מתפריט ראשי כתוצאה מכך נפרש תפריט משנה ש-Windows מרחיבה באופן אוטומטי, תרשים 4.2 מציג תפריט ראשי פשוט ותפריט משנה.

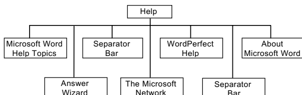


תרשים 4.2: תפריט ראשי פשוט ותפריט משנה.

הערה: אל לך לטעות ולהתבלבל, ולחשוב על תפריטים נשלפים (Pop-Up Menus) בהקשר של תפריטים תלויי הקשר (Context-Sensitive Menus) של Windows API, שספרים רבים מתייחסים אליהם כתפריטים נשלפים. תפריט תלוי הקשר הוא סוג מיוחד של תפריט, אשר נלמד עליו בהמשך.

4.6 מבנה התפריט

Windows בונה את תפריט התוכנית בצורה מדורגת, היררכית. התפריט מורכב מפריט תפריט אחד או יותר; מאפשרות בחירה (Option) אחת או יותר בכל פריט תפריט ראשי; ומבחירה אחת או יותר בכל אפשרות (זהו תפריט-משנה עם אפשרויות משנה, Sub-Options). לדוגמה, תרשים 4.3 מציג מבנה פשוט של עץ תפריטים עבור תוכנית כלשהי. שים לב למבנה ההירכי: בתפריטים הראשיים יש מספר משתנה של אפשרויות, ובחלק מהן אפשרויות בחירה נוספות (תפריטי משנה).



תרשים 4.3: המבנה ההיררכי של תפריטים.

חשוב לשים לב שבמבנה תפריט טוב, יש דרך אחת בלבד כדי להגיע לאפשרות מסוימת, או לבחירה מתוך אפשרויות התפריט. השמירה הקפדנית על צורתו ההיררכית של התפריט מגינה על המשתמש מבלבול בבחירה מתוך אפשרויות משנה (Sub-Options) בתפריטים רבים. כאשר משתמשים במבנה תפריט המעוצב בקפדנות בצורה היררכית, המבנה שלו בתוכנית צריך להיראות כמו בסכימה של תרשים 4.2 ובתרשים 4.3.



תרשים 4.4: התפריט שנראה בתוכנית.

4.7 יצירת תפריט בקובץ המשאבים

ברוב המקרים יוצרים את תפריטי התוכנית בקובץ משאבים. רוב סביבות הפיתוח של C++ עבור Windows מאפשרות ליצור תפריטים על ידי שימוש בטכניקות פשוטות של **גרירה ושחרור**. עם זאת, חשוב להבין איך התוכניות יוצרות תפריטים במקרה שבו חייבים ליצור תפריט באופן עצמאי. להלן קטע מן התוכנית **Menu** (הנמצאת בשלמותה בתקליטור chap04) המציג דוגמה לבניית תפריט בקובץ המשאבים:

```
MYAPP MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit", IDM_EXIT
    END
```

```
    POPUP "&Test!", IDM_TEST
    BEGIN
        MENUITEM "Item &1", IDM_ ITEM1
        MENUITEM "Item &2", IDM_ ITEM2
        MENUITEM "Item &3", IDM_ ITEM3
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About My Application...", IDM_ABOUT
    END
END
```

במספר סעיפים שיבואו בהמשך תבחן בפירוט את ההגדרות שתשתמש בהם להגדרת תפריט בקובץ המשאבים; גם תבחן כיצד להשתמש בהגדרות אלו בתוכנית. בסעיף 4.8 שבהמשך תראה איך משתמשים בקובץ המשאבים ב**מתארים** (Descriptors), או **מילות מפתח** POPUP ו-MENUITEM. חשוב לשים לב לכך שהמתאר DISCARDABLE שראית בדוגמה הקודמת אומר למקשר של המהדר (Compiler's Linker) שיש צורך למחוק את מידע המשאבים המקורי אודות התפריט, לאחר שהתוכנית רושמת את התפריט במחלקת החלון. כמעט תמיד נשתמש במתאר DISCARDABLE עם הגדרות התפריט, כדי לשמור על שטחי הזיכרון ולשפר את מהירות עיבוד התוכניות.

4.8 המתארים POPUP ו-MENUITEM

בסעיף קודם ראית שהחלק המתאר את התפריט בקובץ המשאבים מתחיל בהוראה BEGIN ומסתיים בהוראה END. תיאור תפריט יכול להכיל כל מספר של פריטים מסוג POPUP או מסוג MENUITEM. עם זאת, צריך להיזהר מליצור יותר מדי תפריטים ראשיים, כי ייתכן שלא יהיה לכולם מקום בשורת התפריט, גם כשהחלון בגודלו הרגיל.

בחלק של תיאור התפריט, המתאר POPUP מציין את החלק הראשי, העליון, של התפריט. לדוגמה, בסעיף הקודם ראית שהפריט !Test נמצא בתפריט הראשי של התוכנית. השימוש בתו (&) בתוך המחרוזת מציין שהמשתמש יכול להשתמש **במקש מהיר** (Accelerator Key) בצירוף Alt, ולא בעכבר, כדי להפעיל את הפריט שנבחר בתפריט. פעמים רבות מתייחסים המתכנתים למקש המהיר כ**מקש קיצור של המקלדת** (Keyboard Shortcut). במקרה של הפריט !Test, המקש המהיר הוא Alt+t.

קובץ המשאבים משתמש במתאר MENUITEM כדי להגדיר כל פריט בתוך התפריט. לדוגמה, בתפריט המתאר את !Test בסעיף 4.7 יש שלושה פריטי תפריט: פריט #1, פריט #2 ופריט #3; שים לב שבכל פריט יש התייחסות לקבוע, ועבור פריט #1 הקבוע שמתייחסים אליו הוא IDM_ITEM1. בעזרת הפונקציה WndProc התוכנית תלכוד תחילה את ההודעות מסוג WM_COMMAND. לאחר מכן, מילת הסדר-הנמוך של הערך wParam שהפונקציה WndProc מקבלת, מכילה את הקבוע המזהה את פריט התפריט שנבחר על ידי המשתמש. לדוגמה, כדי לבחור באפשרות IDM_ITEM1, התוכנית

תשתמש בשתי הוראות Switch. ההוראה הראשונה היא לאימות סוג ההודעה כ-WM_COMMAND; ההוראה השנייה בודקת את הערך של wParam כדי לאמת את הפריט התפריט שנבחר על ידי המשתמש היה פריט #1.

לפניך הקוד המתאים:

```
switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_ITEM1 :
                                 // some processing
                 case IDM_ITEM2 :
                                 // more processing
```

4.9 הוספת תפריט לחלון היישום

הגדרת תפריט בקובץ המשאבים (RC) אינה מספיקה להפעלת התפריט, או להצמדת התפריט לחלון היישום. בדרך כלל מצמידים תפריט היישום להגדרת מחלקת החלון שבפונקציה WinMain לפני שקוראים לפונקציה RegisterClass, כמו שראית בסעיפים קודמים. זכור שקבעת בתחילה ערך לשדה lpszMenuName במבנים WNDCLASS ו-WNDCLASSEX, כדי לבצע קישור מוקדם. הפונקציה RegisterClass תשייך עכשיו את שם התפריט לכל חלון שהתוכנית תיצור ממחלקה זו.

באפשרותך גם להשתמש בפונקציות CreateWindow ו-LoadMenu כדי להצמיד את התפריט לחלון. כשקוראים לפונקציות CreateWindow, באפשרותך לקבוע לפרמטר hMenu את הערך שהפונקציה LoadMenu מחזירה. אחר כך תקרא לפונקציה LoadMenu עם שם התפריט מתוך קובץ המשאבים כפרמטר של הפונקציה, כמו שתראה להלן:

```
if (LOWORD(wParam) == IDM_NEW )
  hNewMenu = LoadMenu(hInst, "NEWMENU");
else
  hNewMenu = LoadMenu(hInst, "OLDMENU");
```

לבסוף, באפשרותך להשתמש בפונקציות SetMenu ו-LoadMenu להצמדת תפריט לחלון אחרי שתיצור את החלון. עליך לקרוא לפונקציה SetMenu עם הידית hMenu כפרמטר. SetMenu מקשרת את התפריט שהידית hMenu מצביעה עליו אל חלון היישום שפתוח כרגע, כפי שמוצג להלן:

```
SetMenu(hWnd, hNewMenu);
```

4.10 שינוי תפריטים בתוך היישום

בסעיפים קודמים למדת להשתמש בשיטות שונות כדי להצמיד תפריטים ליישום. אך מעבר לעניין תכנון תוכניות Windows, אתה עשוי לפתח יישומים מורכבים יותר שנדרש בהם שינוי של התפריט כשהיישום מפעיל. ממשק התכנות Win32 API מספק אוסף של פונקציות שאפשר להשתמש בהן בתוכניות, כדי לשנות תפריט בזמן פעולת היישום. חלק מהפונקציות האלו נמצאות בנספח 1. ככלל, הפונקציות מאפשרות לשנות כל אלמנט תפריט לאחר שהצמדת את התפריט לחלון.

4.11 הודעות שנוצרות על ידי תפריטים

בכל פעם שהמשתמש בוחר באחד מפריטי תפריט כלשהו, Windows שולחת את ההודעה WM_COMMAND אל לולאת ההודעות של התוכנית. בלולאת ההודעות שבתוכנית צריך לבדוק את מילת הסדר-הנמוך של הערך wParam (זכור, הערך שמכיל הפרמטר wParam הוא מסוג DWORD), כדי לקבוע איזה פריט תפריט נבחר.

ככלל, ההודעה WM_COMMAND היא היחידה ש-Windows שולחת לתוכניות כתוצאה מבחירת פריט בתפריט התוכנית. אם המשתמש בוחר פריט מתפריט המערכת, למשל, Windows תשלח במקום הודעה זו את ההודעה WM_SYSCOMMAND (שקרוב לוודאי תטפל בה על ידי הפונקציה DefWindowProc). ייתכן שהיישום שלך ידרוש מלולאת ההודעות שתטפל בהודעות WM_INITMENU ו-WM_INITMENUPOPUP, אשר נשלחות על ידי המערכת לחלון מייד לפני שהמערכת מפעילה את התפריט (תפריט ראשי או תפריט נשלף, בהתאם להודעה). תפיסת ההודעות WM_INITMENU ו-WM_INITMENUPOPUP מאפשרת ליישום לשנות תפריט או תפריטים, אם תצטרך לעשות זאת, מייד לפני ש-Windows מציגה את תפריט היישום.

התפריט ישלח את ההודעה WM_MENUSELECT בכל פעם שהמשתמש בוחר בו פריט כלשהו. הודעה זו מתוחכמת יותר מההודעה WM_COMMAND, מכיון ש-Windows יוצרת אותה אפילו אם פריט התפריט אינו פעיל (disabled). עם זאת, בדרך כלל משתמשים בהודעה WM_MENUSELECT רק כדי להציג תפריט עזרה תלוית-הקשר. הפעל את התוכנית **Menu** הנמצאת בתקליטור.

4.12 הוספת מקשים מהירים

תכונה אחרונה שנדון בה לפני שנניח לנושא התפריטים, היא מקשים מהירים. **מקש מהיר** (Accelerator Key) הוא מקש שאתה מגדיר, ואשר הקשה עליו גורמת לבחירה אוטומטית באפשרות מסוימת מתפריט מסוים, אף על פי שאותו תפריט כלל אינו מוצג על המסך. במילים אחרות, תוכל לבחור פריט נתון במישרין, על ידי הקשה על מקש מהיר, ולעקוף לחלוטין את התפריט. המונח **מקש מהיר** מיטיב לתאר תכונה זו, מכיון

שבדרך כלל משך הזמן הדרוש לבחירה בפריט מתפריט באמצעות מקש מהיר, קצר מזה הנדרש כשאתה קודם מפעיל את התפריט ואחר כך בוחר ממנו את הפריט הרצוי. מקובל גם המונח **מקש קיצור** (Shortcut Key), במקום מקש מהיר.

כדי להגדיר מקשים מהירים עבור תפריט נתון, עליך להוסיף טבלת מקשים מהירים לקובץ המשאבים של התפריט. המתכונת הכללית לכל הגדרות טבלאות המקשים המהירים היא:

```
TableName ACCELERATORS
{
  Key1, MenuID1 [,type] [option]
  Key2, MenuID2 [,type] [option]
  Key3, MenuID3 [,type] [option]
  .
  .
  .
  Keyn, MenuIDn [,type] [option]
}
```

בדוגמה, TableName הוא השם של טבלת המקשים המהירים. הפרמטר Key מכיל את המקש הגורם לבחירת הפריט, ואילו MenuID הוא ערך ID הקשור בפריט הרצוי. הפרמטר type מציין אם המקש הוא מקש רגיל (ברירת המחדל), או מקש וירטואלי (נדון בנושא זה בהמשך). האפשרויות לבחירה הן אחת מפקודות המאקרו הבאות: NOINVERT, ALT, SHIFT ו-CONTROL. NOINVERT מונע הדגשה של פריט התפריט הנבחר בעת הלחיצה על המקש המהיר. ALT מציין את מקש Alt, SHIFT מציין את מקש Shift, ו- CONTROL מציין את המקש Ctrl.

הערך של הפרמטר Key יהיה תו בתוך גרשיים, ערך מספרי המקביל להגדרת המקש כ-ASCII, או קוד למקש וירטואלי. אם תשתמש בתו נתון בגרשיים, המערכת תניח שמדובר בתו ASCII. אם השתמשת בערך מספרי, עליך להודיע במפורש למהדר המשאבים שזהו תו ASCII, על ידי סימון ASCII עבור הפרמטר type. אם זהו מקש וירטואלי, יש לכתוב VIRTKEY במקום type.

אם המקש הוא אות רישית בתוך גרשיים, פריט התפריט המתאים ייבחר כאשר המשתמש יקיש על אותו מקש, בעת שמקש Shift לחוץ. אם זהו תו רגיל, הפריט המתאים ייבחר בעקבות הקשה על אותו מקש בלבד. אם המקש הוגדר כתו רגיל, עם אפשרות Alt, אזי לחיצה על Alt והקשה על אותו מקש תגרום לבחירת הפריט המתאים (אם המקש מיוצג על ידי אות רישית בצירוף Alt, אזי יש ללחוץ על Shift ו- Alt יחד עם אותו מקש כדי לבחור את הפריט המתאים). אך אם ברצונך שהמשתמש ילחץ על מקש Ctrl ועל האות כדי לבחור בפריט מסוים, הוסף לפני שם המקש את התו ^ (למשל, A^).

מקש וירטואלי (Virtual Key) הוא קוד שאינו תלוי-מערכת. בין המקשים הווירטואליים נמנים מקשי הפונקציות F1 עד F12, מקשי החיצים, ומקשים נוספים שאין להם הגדרה בטבלת ASCII. מקשים אלה מוגדרים באמצעות פקודות מאקרו

בקובץ הכותר windows.h (או באחד מנגזרותיו). כל פקודות המאקרו למקשים וירטואליים מתחילות ב- _VK. לדוגמה, פקודות המאקרו למקשי הפונקציות הם VK_F1 עד VK_F12. עליך לעיין בקובץ windows.h כדי לברר מהם קודי המאקרו עבור שאר המקשים הווירטואליים. כדי להפוך מקש וירטואלי למקש מהיר, עליך לציין בפרמטר key את המאקרו המגדיר אותו, ובפרמטר type הקלד VIRTKEY. כמובן שתוכל לציין גם ALT, SHIFT או CONTROL כדי להגדיר את צירוף המקשים הרצוי.

להלן מספר דוגמאות:

```
"A", IDM_x ; select by pressing Shift-A
"a", IDM_x ; select by pressing a
"^A", IDM_x ; select by pressing Ctrl-A
"a", IDM_x, ALT ; select by pressing Alt-A
VK_F2, IDM_x ; select by pressing F2
VK_F2, IDM_x, SHIFT ; select by pressing Shift-F2
```

לפניך חלק מקובץ המשאבים Accel.rc בתוספת הגדרות למקשים מהירים עבור התפריט שפיתחנו בסעיף הקודם:

```
MYMENU ACCELERATORS MOVEABLE PURE
BEGIN
    "G",            IDM_GAMMA,              ASCII,  NOINVERT
    VK_F1,          IDM_ABOUT,              VIRTKEY, NOINVERT
    "^E",           IDM_ITEM1,              ASCII,  NOINVERT
    "^Z",           IDM_ITEM2,              ASCII,  NOINVERT
    VK_F4,          IDM_ITEM3,              VIRTKEY, CONTROL,
NOINVERT
END
```

שים לב שהגדרת התפריט הורחבה, כך שיוצגו בה הקישורים בין מקשים מהירים לבין האפשרויות בתפריט. כל פריט מופרד מהמקש המהיר שלו בטאב. קובץ הכותר windows.h נכלל כאן, מכיוון שהוא מגדיר את פקודות המאקרו של המקשים הווירטואליים.

4.13 כיצד לטעון את טבלת המקשים המהירים

אף שהגדרות המקשים המהירים שמורות באותו קובץ משאבים כמו התפריט, יש לטעון אותן בנפרד בעזרת פונקציית API אחרת, הנקראת ()LoadAccelerators, שהגדרתה היא:

```
HACCEL LoadAccelerators(HINSTANCE ThisInst, LPCSTR Name);
```

הפרמטר ThisInst הוא הידית ליישום, ואילו Name מכיל את שם טבלת המקשים המהירים. הפונקציה מחזירה ידית לטבלת המקשים המהירים, או ערך NULL אם לא ניתן לטעון את הטבלה.

עליך להפעיל את הפונקציה ()LoadAccelerators מייד לאחר שהחלון נוצר. הדוגמה הבאה מראה כיצד לטעון את טבלת המקשים המהירים של MYMENU :

```
HACCEL hAccel;
hAccel = LoadAccelerators(hThisInst, "MYMENU");
```

הערך שמכיל הפרמטר hAccel ישמש אותנו אחר כך ויסייע לטפל במקשים מהירים.

למרות שהפונקציה ()LoadAccelerators טוענת את טבלת המקשים המהירים, תוכל התוכנית שלך לטפל בהם (לבצע את הוראתם), לאחר שתוסיף פונקציית API נוספת ללולאת ההודעות. פונקציה זו נקראת ()TranslateAccelerator, והגדרתה היא :

```
int TranslateAccelerator(HWND hwnd, HACCEL hAccel, LPMSG lpMess);
```

במקרה זה, מכיל הפרמטר hwnd ידית לחלון שעבורו יתורגמו המקשים המהירים. hAccel הוא הידית לטבלת המקשים המהירים שתשמש לצורך זה. זוהי הידית המוחזרת על ידי הפונקציה ()LoadAccelerators. ולבסוף, lpMess הוא מצביע, המצביע על ההודעה. הפונקציה ()TranslateAccelerators מחזירה ערך true אם היתה לחיצה על מקש מהיר, וערך false אם לא. הפונקציה מתרגמת כל הקשה על מקש מהיר להודעת WM_COMMAND מתאימה.

כאשר אתה משתמש בפונקציה ()TranslateAccelerators, על לולאת ההודעות שלך להיראות כך :

```
while(GetMessage(&msg, NULL, 0, 0))
{
  if(!TranslateAccelerator(hwnd, hAccel, &msg)) {
    TranslateMessage(&msg); /* allow use of keyboard */
    DispatchMessage(&msg); /* return control to Windows */
  }
}
```

הפונקציה TranslateAccelerator מתרגמת כל הודעה של טבלת מקשים מהירים (AcceleratorTable message) להודעה מתאימה מסוג WM_COMMAND ולשילוב של קבועי תפריט. שים לב שהתוכנית בודקת את הפונקציה TranslateAccelerator. כאשר פונקציה זו מטפלת בהודעה, הלולאה לא צריכה לאפשר את המשך הטיפול הרגיל, מכיוון ש-TranslateAccelerator קוראת בצורה אוטומטית לפונקציה WndProc.

כדי לנסות את השימוש במקשים מהירים, תמצא בתקליטור את התוכנית **Accel**. (הבאנו פה רק את הפונקציה ()WinMain של התוכנית. התוכנית בשלמותה נמצאת בתיקיה: Chap04\Accel):

```
int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                    hPrevInstance,LPTSTR lpCmdLine, int
nCmdShow)
{
   MSG       msg;
   HWND      hWnd;
   WNDCLASS wc;
   HACCEL hAccel;

    // Register the main application window class.
   //.........................................
   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = (WNDPROC)WndProc; /* window function */
   wc.cbClsExtra     = 0;
   wc.cbWndExtra     = 0;
   wc.hInstance      = hInstance;  /* handle to this instance */
   wc.hIcon          = LoadIcon( hInstance, lpszAppName );
                       /* icon style */
   wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
                       /* cursor style */
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName; /* window class name */

   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );

    hInst = hInstance;
```

```
   // Create the main application window.
   //....................................
   hWnd = CreateWindow( lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                      );

   if ( !hWnd )
      return( FALSE );


   hAccel = LoadAccelerators(hInst, "MYMENU");

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );

   while( GetMessage( &msg, NULL, 0, 0) )
   {
      if(!TranslateAccelerator(hWnd, hAccel, &msg)) {
         TranslateMessage( &msg );
         DispatchMessage( &msg );
      }
   }

   return( msg.wParam );

}
```

לפני שתמשיך לפרק הבא, רצוי שתערוך כמה ניסיונות בתכנות תיבות הודעה, תפריטים ומקשים מהירים. נסה את האפשרויות השונות ובדוק מה הן עושות. אנו נשתמש בתפריטים ובתיבות הודעה במרבית התוכניות שנציג בספר זה, ועל כן חשוב שתרכוש הבנה יסודית בנושא זה.

פרק **5**

היכרות עם תיבות דו-שיח

כשנתבונן היטב נראה שתיבת דו-שיח דומה לחלון נשלף (או חלון קופץ). היא מקבלת קלט מהמשתמש למשימה מסוימת, כמו למשל, שם קובץ או מחרוזת תווים לחיפוש. ההבדל העקרוני בין תיבות דו-שיח לחלונות נשלפים הוא, שתיבות דו-שיח משתמשות ב**תבניות** (Templates) שמגדירות את **הפקדים** (Controls) שתיבת הדו-שיח מציגה. משתמשים בסוג המשאב DIALOG כדי להגדיר תבניות אלו בקובץ המשאבים. באפשרותך ליצור את התבניות גם באופן דינמי בזיכרון בזמן פעולת התוכנית.

כמו כן, תיבות הדו-שיח פועלות באופן שונה מחלונות נשלפים בכך שהן משתמשות בפונקציית ברירת מחדל מיוחדת לטיפול בהודעות שמפענחות את הקשות המקלדת, כמו מקשי החיצים ומקש הטבלר (Tab Key), אשר מאפשרים למשתמש לבחור בקלות בפקדים שבתיבת הדו-שיח. ככלל, הפונקציה הנוספת לטיפול בהודעות וההצגה של תיבת הדו-שיח עושים את תיבת הדו-שיח לכלי המקובל ביותר לקלט פשוט מהמשתמש ועיבודו.

תיבות דו-שיח (ואמצעי הבקרה הכלולים בהם) הן נושא רחב. בפרק זה תלמד את יסודות הניהול של תיבות דו-שיח, כיצד ליצור אותן וכיצד לעבד הודעות שמקורן בתיבות דו-שיח. בפרקים הבאים, תשתמש בתיבת דו-שיח כדי לחקור כמה ממרכיבי הממשק של חלונות 9x.

5.1 כיצד מתבצעת התקשורת בין תיבת הדו-שיח והמשתמש

תיבת הדו-שיח (Dialog Box) מנהלת את התקשורת עם המשתמש באמצעות פקדים (Controls). פקד הוא סוג מיוחד של חלון לקלט, או פלט. פקדים נמצאים "בבעלות" חלון-האב שלהם; בדוגמאות המובאות בפרק זה, חלון-האב הוא תיבת הדו-שיח.

הדוגמאות בפרק זה מתייחסות לשלושה מהפקדים האלה: לחצנים, תיבות רשימה, ותיבות עריכה. בהמשך הספר, נבחן את שאר סוגי הפקדים.

חשוב להבין שהפקדים האלה מפיקים הודעות (עם כל גישה מצד המשתמש) אך גם מקבלים הודעות (מהיישום שלך). הודעה שהופקה על ידי פקד מציינת את סוג האינטראקציה שהתחוללה בין המשתמש לפקד. הודעה הנשלחת לפקד מסוים היא למעשה, הוראה שהפקד יגיב עליה בפעולה כלשהי. בהמשך הפרק יובאו דוגמאות להעברת הודעות מהסוג שתואר לעיל.

5.2 הגדרת סוגי תיבות דו-שיח

בסעיף הקודם למדת שבאפשרותך להשתמש בתיבות דו-שיח בתוכנית כדי לספק למשתמש מידע, וגם לקבל ממנו מידע. יש שני סוגים עיקריים של תיבות דו-שיח: תיבות דו-שיח **מודאליות** (Modal Dialog Boxs) ותיבות דו-שיח **לא-מודאליות** (Modaless Or Non-Modal Dialog Boxes).

כשתיבת דו-שיח מודאלית מוצגת על המסך, המשתמש **אינו** יכול להמשיך ביישום עד אשר הוא יסגור את התיבה, ובדרך כלל - יקליד נתונים ויאשר, או ילחץ בלחצן כלשהו כנדרש. תיבת דו-שיח מודלית מגבילה את הגישה לשאר החלונות הנראים של היישום שקרא לתיבת הדו-שיח. עם זאת, המשתמש יכול לעבור ליישומים אחרים בזמן שיישום אחד מציג את תיבת דו-שיח מודאלית. תיבת הדו-שיח הפשוטה ביותר מסוג זה היא **תיבת ההודעה** (Message Box), שהשתמשת בה בפרקים הקודמים.

התוכנית יכולה להגדיר גם **תיבת דו-שיח מודאלית למערכת** (System Modal Dialog Boxes). תיבת דו-שיח מודאלית למערכת משתלטת על כל המסך, ואינה מאפשרת למשתמש לבצע עיבוד כלשהו בכל התכניות האחרות עד שהמשתמש יגיב לתיבת הדו-שיח. ההיגיון לשימוש בתיבת דו-שיח מודאלית למערכת הוא רק למקרה של בעיה רצינית שהמשתמש אינו יכול להתעלם ממנה, כמו למשל, שגיאת מערכת. אפשר ליצור תיבת דו-שיח מודאלית למערכת בשתי שיטות. הראשונה - על ידי הגדרת הסגנון **WS_SYSMODAL** בתבנית של תיבת הדו-שיח והשנייה - על ידי שימוש בפונקציה **SetSysModalWindow** כדי ליצור את תיבת הדו-שיח עצמה.

5.3 קבלת הודעות מתיבות דו-שיח

תיבת דו-שיח היא למעשה חלון (אף שזהו חלון מסוג מיוחד). אירועים המתרחשים בתוך תיבת דו-שיח נשלחים אל התוכנית שלך בעזרת מנגנון להעברת הודעות המשמש גם את החלון הראשי. אולם הודעות מתיבת דו-שיח אינן נשלחות לפונקציית החלון של החלון הראשי. במקום זה, כל תיבת דו-שיח שתגדיר תזדקק לפונקציית חלון משל עצמה, שנהוג בדרך כלל לכנותה **פונקציית דו-שיח** (Dialog Function). הגדרת פונקציה זו צריכה להיות כך (מובן ששם הפונקציה נתון לבחירתך):

```
BOOL CALLBACKFunc(HWND hdwnd, UINT message,
                        WPARAM wParam, LPARAM lParam);
```

כפי שאתה רואה, פונקציית דו-שיח מקבלת את אותם פרמטרים כמו פונקציית החלון של החלון הראשי. עם זאת, היא שונה מפונקציית החלון הראשית בכך שהיא מחזירה ערך true או false. בדומה לפונקציית החלון של החלון הראשי, פונקציית חלון של תיבת דו-שיח תקבל הודעות רבות. אם הפונקציה מעבדת הודעה מסוימת, היא תחזיר ערך true. אם היא אינה מגיבה להודעה, היא תחזיר ערך false.

ככלל, כל פקד הכלול בתיבת דו-שיח יקבל מספר זיהוי משלו. כל פעם שהמשתמש מפעיל פקד נשלחת הודעה לפונקציית הדו-שיח, ובה יצוין מספר הזיהוי של אמצעי הבקרה וסוג הפעולה שנקט המשתמש. הפונקציה תפענח את ההודעה ותנקוט את הפעולה המתאימה. תהליך זה זהה לדרך שבה מפענחת פונקציית החלון הראשית של התוכנית את ההודעות שהיא מקבלת.

שלא כמו תיבת הדו-שיח המודאלית, שזמן פעולתה מוגדר, באפשרות תיבת הדו-שיח הלא-מודאלית לקבל וגם לאבד את **מיקוד הקלט** (Input Focus), כלומר, תיבת דו-שיח לא מודאלית יכולה להיות חלון פעיל או להיות חלון לא פעיל (כמו לדוגמה, בעת חיפוש במעבד התמלילים Word). לתיבת דו-שיח לא-מודאלית יש זמן פעולה לא מוגדר. מכיוון שתיבת דו-שיח לא-מודאלית יכולה להיות חלון לא פעיל, התוכניות שמשתמשות בה חייבות לוודא שניתנת האפשרות גם לתיבת הדו-שיח לקבל את ההודעות שנשלחות אליה. בדרך כלל תבנה לולאת הודעות עבור תוכניות שמכילות תיבות דו-שיח לא-מודאליות, כמו שבדוגמה שלהלן:

```
while(GetMessage(&msg, NULL, 0, 0) )
  {
    if (hDlgModeless || IsDialogMessage(hDlgModeless, &msg))
      {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
      }
 }
```

בצורת הבניה המקובלת, הערך hDlgModaless מציין ידית לתיבת הדו-שיח הלא-מודאלית. אם תיבת הדו-שיח הלא-מודאלית אינה פתוחה כרגע, הערך hDlgModaless חייב להיות NULL. הפונקציה IsDialogMessage קובעת אם הודעה מ-Windows מיועדת לתיבת הדו-שיח. אם כן, הפונקציה תשלח את ההודעה אל פונקציית תיבת הדו-שיח שמטפלת בהודעות, ולכן הפונקציה TranslateMessage והפונקציה DispatchMessage לא יטפלו בהודעה.

5.4 כיצד ליצור תיבת דו-שיח פשוטה

תיבת הדו-שיח הראשונה שניצור תהיה דוגמה פשוטה שתכיל שלושה לחצנים: Red, Green ו-Cancel. לחיצה על לחצן Red, או על Green תגרום להפעלת תיבת הודעה שתציין איזה לחצן נבחר. התיבה תעלם מהמסך בעקבות לחיצה על לחצן Cancel.

בדוגמה זו ובשאר הדוגמאות בפרק זה, לא נעשה שימוש רב במידע שמעבירה תיבת הדו-שיח. עם זאת, יש בהן כדי להדגים היטב את התכונות העיקריות של תיבות הדו-שיח שישמשו אותך ביישומים שתכתוב.

5.5 קובץ המשאבים של תיבת הדו-שיח

גם תיבות הדו-שיח הן משאבים הכלולים בקובץ המשאבים של התוכנית. לפני שתפתח תוכנית העושה שימוש בתיבות דו-שיח, עליך לבנות קובץ משאבים המגדיר תיבות דו-שיח.

רכיבי תבנית תיבת הדו-שיח

הגדרת תיבת דו-שיח מתחילה בהוראה DIALOG, שנכתבת לפני סדרת משאבי הפקדים של תיבת הדו-שיח. המבנה הכללי של תבנית תיבת הדו-שיח דומה לתבנית המשאב שהשתמשת בה בסעיפים קודמים, כדי לתכנן תפריטים ופריטי תפריטים. המבנה הכללי של תבנית תיבת הדו-שיח:

```
DBIdentifier DIALOG DISCARDABLE  Left, Top, Width, Height
STYLE DS_ Style1 | Style2 | ... | StyleN
CAPTION "Dialog "
FONT Font size, "FontName"
BEGIN
ControlType1 "Control caption", Control_ID, Left, Top,
Width, Heigh
ControlType1 "Control caption", Control_ID, Left, Top,
Width, Heigh
 ...
ControlType1 "Control caption", Control_ID, Left, Top,
Width, Heigh
END
```

DBIdentifier הוא שם תיבת הדו-שיח. הפינה השמאלית-העליונה של התיבה תהיה בנקודה Left, Top. מימדי התיבה נתונים על ידי הפרמטר Width והפרמטר Height.

טבלה 5.1: סגנונות נפוצים לתיבות דו-שיח

הערך	הפעולה
DS_MODALFRAME	יוצר תיבת דו-שיח מודאלית
WS_BORDER	כוללת קו מתאר
WS_CAPTION	כוללת פס כותרת
WS_CHILD	יוצר תיבת דו-שיח בסגנון חלון-בן
WS_POPUP	יוצר תיבת דו-שיח בסגנון חלון קופץ
WS_MAXIMIZEBOX	כולל לחצן הגדלה
WS_MINIMIZEBOX	כולל לחצן הקטנה
WS_SYSMENU	כולל תפריט מערכת
WS_TABSTOP	בחירת פקד בלחיצה על Tab
WS_VISIBLE	התיבה נראית כשהיא פעילה

ההגדרה ControlType מתייחסת לאחד מסוגי חלונות הבן שאפשר להשתמש בהם כדי ליצור פקדים בתיבת דו-שיח. ההגדרה ControlType חייבת לכלול את אחד הערכים שברשימה המפורטת בטבלה 5.2.

טבלה 5.2: הערכים האפשריים עבור הכניסה ControlType בהגדרת תיבת הדו-שיח.

סוגי פקדים אפשריים			
BUTTON	CHECKBOX	COMBOBOX	CONTROL
CTEXT	DEFPUSHBUTTON	EDITTEXT	GROUPBOX
ICON	LISTBOX	LTEXT	PUSHBUTTON
RADIOBUTTON	RTEXT	SCROLLBAR	STATIC

לפניך תיאור קצר של הסוגים:

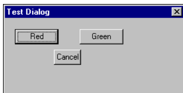
- **לחצן** (push button) הוא פקד שהמשתמש "לוחץ עליו" (על ידי לחיצה בעכבר, או הקשה על מקש Tab עד שהלחצן הרצוי מואר, ולאחר מכן הקשה על Enter) כדי לחולל תגובה מסוימת. לדוגמה, לחצן OK שבו אנו משתמשים ברוב תיבות ההודעה, הוא פקד מסוג "לחצן".
- **תיבת סימון** (Check Box) מכילה פריט אחד או יותר, ולצידם סימן "°" (או אחר), או מקום ריק. אם הפריט מסומן, פירוש הדבר שנבחר. אם תיבת דו-שיח מכילה יותר משתי תיבות סימון, ניתן לבחור ביותר מפריט אחד.

- ☆ **כפתור רדיו** (Radio Button) הוא בעיקרו של דבר תיבת סימון. ההבדל הוא, שניתן לבחור באפשרות אחת בלבד בכל פעם, ללא קשר למספר הכפתורים המוצגים. כפתור רדיו הוא תיבת סימון המוציאה מכלל אפשרות (Exclusive) כל תיבת סימון אחרת.
- ☆ **תיבת רשימה** (List Box) היא תיבה ובה רשימת פריטים, שהמשתמש רשאי לבחור באחד מהם (או ביותר). תיבות רשימה משמשות בדרך כלל להצגת פריטים, כמו שמות קבצים.
- ☆ **תיבת עריכה** (Edit Box) מאפשרת למשתמש להקליד מחרוזת תווים. תיבות אלו מכילות את כל המאפיינים הנחוצים לעריכת טקסט. לכן, כשהתוכנית מבקשת מחרוזת תווים, היא מציגה לפני המשתמש תיבת עריכה ומחכה עד שיסיים להקליד את המחרוזת הרצויה לו.
- ☆ **תיבה משולבת** (Combination Box) היא צירוף של תיבת רשימה ותיבת עריכה.
- ☆ **פס גלילה** (Scroll Bar) משמש לגלילת הטקסט בחלון.
- ☆ **פקד סטטי** (Static Control) משמש ליצירת פלט טקסט (או גרפיקה) מהמשתמש, אך אינו יכול לקבל קלט.

יצירת תבנית לתיבת דו-שיח

בסעיף הקודם למדת על הגדרת המבנה הכללי של תבנית תיבת הדו-שיח, שכללה מספר מרכיבים חשובים. ייתכן, שיהיה לך יותר קל להבין את הגדרת תיבת הדו-שיח, אם תנתח הגדרה של תיבת דו-שיח פשוטה, כמו בדוגמה שלהלן (דוגמה זו תשמש אותנו ליצור תיבת דו-שיח ראשונה):

```
TESTDIALOG DIALOG DISCARDABLE  20, 20, 180, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
      WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
{
    DEFPUSHBUTTON   "Red",IDD_RED,10,10,44,14,
            WS_CHILD | WS_VISIBLE
    PUSHBUTTON        "Green",IDD_GREEN,75,10,44,14
            WS_CHILD | WS_VISIBLE | WS_TABSTOP
    PUSHBUTTON        "Cancel",IDCANCEL,49,29,27,14
            WS_CHILD | WS_VISIBLE | WS_TABSTOP
}
```



ההגדרות שלעיל יוצרות את תיבת הדו-שיח שמוצגת בתרשים 5.1.

כשמסתכלים מקרוב על תיבת הדו-שיח שנוצרה, קל מאוד להבין את הרכיבים שלה המפורטים בקובץ המשאבים המוגדר. כל הוראה שבבלוק ההגדרות Begin-End יוצרת פקד נפרד. הראשון הוא לחצן ברירת המחדל. לחצן זה מואר אוטומטית כאשר תיבת הדו-שיח מוצגת. לפניך התצורה הכללית של הכרזה על לחצן:

```
PUSHBUTTON "string", PBID, X, Y, Width, Height [, Style]
```

בדוגמה זו מכיל הפרמטר String את הטקסט שיופיע על הלחצן. PBID הוא הערך הקשור אל הלחצן. זהו הערך המוחזר אל התוכנית שלך עם כל לחיצה על הלחצן. הפינה השמאלית-העליונה של הלחצן תמוקם בנקודה X,Y, וגודלו של הלחצן יהיה על פי ההגדרות בפרמטרים Width ו- Height. הפרמטר Style קובע את אופיו המדויק של הלחצן. כדי להגדיר לחצן בעל סגנון ברירת מחדל, השתמש במשפט DEFPUSHBUTTON. למשפט זה פרמטרים זהים לאלה של הלחצנים הרגילים (רק שהערכים WS_TABSTOP ו- BS_PUSHBUTTON כבר כלולים בו, כך שנחסך הצורך להגדיר את סוג הפקד והפרמטרים שלו).

ההוראה האחרונה בבלוק משתמשת במזהה IDCANCEL כדי ליצור את הלחצן Done. בדרך כלל, מציבים את המזהה IDCANCEL לכל לחצן שאמור לסגור את תיבת הדו-שיח, מבלי לשמור שינויים במקומות אחרים.

הגדרת הרכיבים של תיבת הדו-שיח

לפני פירוט הגדרות הפקד, תיבת הדו-שיח מגדירה מאפיינים סטנדרטיים משל עצמה. לדוגמה, תיבת הדו-שיח TESTDIALOG בקטע הקודם, מתחילה עם חמש השורות הבאות:

```
TESTDIALOG DIALOG DISCARDABLE  20, 20, 180, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
      WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
```

השורה הראשונה מציינת את תיבת הדו-שיח עם המזהה TESTDIALOG. בשורה גם מוגדרים המידות של תיבת הדו-שיח. בדוגמה הקודמת, תיבת הדו-שיח מתחילה ב- 20 יחידות DBU למטה (Dialog Based Units, יחידת מידה בתיבת דו-שיח להגדרת מיקום וגודל) ו- 20 יחידות DBU משמאל לשפה השמאלית של שטח הלקוח בחלון הקורא (כלומר, הפינה השמאלית-העליונה של התיבה תהיה בנקודה [20,20] יחסית לשטח הלקוח של החלון). רוחבה של תיבת הדו-שיח הוא 180 יחידות DBU וגובהה - 70 יחידות DBU.

השורה השנייה והמשכה בשורה השלישית, מגדירות את הסגנונות שתיבת הדו-שיח משתמשת בהם כשהיא נוצרת. באפשרותך להשתמש בסגנונות החלון, אלה שמתחילים ב-WS או DS כסגנון של תיבת דו-שיח. צריך לכלול תמיד את הסגנון WS_VISIBLE כדי לגרום להצגת תיבת הדו-שיח, ואי אפשר להשתמש בסגנונות WS_MINIMIZEBOX או WS_MAXIMIZEBOX.

משתמשים במילת המפתח CAPTION, כמו שרואים בשורה הרביעית, עם תיבות דו-שיח שכוללות בהגדרת הסגנון שלהן את הסגנון WS_CAPTION. ככלל, צריך לתת כותרת לתיבות הדו-שיח משתי סיבות: ראשית, הכותרת מזכירה למשתמש את מטרת תיבת הדו-שיח; ושנית, היא מאפשרת למשתמש להזיז את תיבת הדו-שיח בתחום המסך. הקפד לכתוב את המחרוזת של כותרת תיבת הדו-שיח בין גרשיים.

לבסוף, מילת המפתח FONT אינה מגדירה רק את **צורת הגופן** (Typeface) שתיבת הדו-שיח משתמשת בו, אלא גם את גודל כל פקד בתיבת הדו-שיח ואת תיבת הדו-שיח עצמה. למילת המפתח FONT תפקיד חשוב בקביעת גודלן של תיבות הדו-שיח, מכיוון ש-Windows מחשבת את יחידות ה-DBU כחלק יחסי מגודל הגופן. עבור רוב תיבות הדו-שיח, גופן בגודל 8 נקודות, כמו MS San Serif, זו בחירה טובה. חייבים לוודא ששם הגופן שנכתב בתוך הגרשיים זהה לשם הגופן שמוגדר במערכת; אחרת, קובץ המשאבים לא יעבור את ההידור כנדרש.

הגדרת הפקדים של תיבת דו-שיח

```
CONTROL "Push if You're Happy", IDC_BUTTON1, BUTTON,
   BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 45, 66, 48, 12
DEFPUSHBUTTON "Push if You're Happy", IDC_BUTTON1,
   45, 66, 48, 12
```

שתי ההגדרות מקובלות; אך, כמו שלמדת כבר בדוגמאות הרבות שהיו לך עד כה, עליך לבחור באחת השיטות, זו הנוחה לך יותר, ולהתמיד בשימוש בה בקובץ המשאבים.

עבור הפקדים עם פרמטר הרשות Style, הבחירות כוללות את הסגנונות WS_TABSTOP ו-WS_GROUP. הסגנונות WS_TABSTOP ו-WS_GROUP שולטים בפעולת ברירת המחדל של ממשק המקלדת, כמו שמוסבר בסעיף 5.8. צריך להשתמש באופרטור OR (|) הפועל על סיביות (Bitwise OR) כדי לשלב בין כל הסגנונות שאתה קובע לפקד.

5.6 הצגת תיבת דו-שיח עם המאקרו DialogBox

למדת שמגדירים בקובץ המשאבים את תבנית תיבת הדו-שיח. בסעיף זה ובמספר סעיפים אחריו, תלמד על שיטות שונות שאפשר להשתמש בהן להצגת תיבות דו-שיח. מבין כל השיטות האפשריות, השיטה הפשוטה ביותר היא באמצעות המאקרו DialogBox. מאקרו זה יוצר תיבת דו-שיח מודאלית ממשאב של תבנית תיבת דו-שיח. DialogBox אינו מחזיר שליטה לתוכנית שקראה לה, עד שפונקציית המשוב המוגדרת מסיימת את תיבת הדו-שיח המודאלית על ידי קריאה לפונקציה EndDialog (תלמד יותר על הפונקציה EndDialog בסעיף 5.14). המאקרו DialogBox משתמש בפונקציה DialogBoxParam, שדומה לפונקציה CreateDialogParam המוסברת בסעיף 5.12 בפירוט. ההגדרה של המאקרו DialogBox היא כמו בדוגמה זו:

```
int DialogBox(
   HINSTANCE, hInstance,  // handle to application instance
   LPCTSTR lpTemplate,    // identifies dialog box template
   HWND hWndParent,       // handle to owner window
   DLGPROC lpDialogFunc // pointer to dialog box procedure
);
```

המאקרו DialogBox מקבל ארבעה פרמטרים, כמפורט בטבלה 5.3.

טבלה 5.3: הפרמטרים של הפונקציה **DialogBox**.

פרמטר	תיאור
hInstance	מזהה את מופע התוכנית שקובץ ההפעלה שלה מכיל את תיבת הדו-שיח.
lpTemplate	מזהה את תבנית תיבת הדו-שיח. פרמטר זה יכול להיות המצביע למחרוזת תווים המסתיימת ב-NULL, שמגדירה את שם תבנית תיבת הדו-שיח; או שהוא יכול להיות ערך של מספר שלם המגדיר את מזהה המשאב של תבנית תיבת הדו-שיח. כאשר הפרמטר מגדיר מזהה משאב, מילת הסדר-הגבוה חייבת להיות שווה לאפס, ומילת הסדר-הנמוך חייבת להכיל את המזהה. באפשרותך להשתמש במאקרו MAKEINTRESOURCE כדי ליצור ערך זה.
hWndParent	מזהה את חלון האב של תיבת הדו-שיח.
lpDialogFunc	מצביע לפונקציית החלון של תיבת הדו-שיח.

אם הקריאה ל-DialogBox מסתיימת בהצלחה, הפונקציה מחזירה את הפרמטר nResult. התוכנית תשתמש באופן עקבי בפרמטר nResult בקריאה לפונקציה EndDialog שסוגרת את תיבת הדו-שיח. אם הקריאה ל-DialogBox אינה מצליחה, הפונקציה מחזירה את הערך 1-.

המאקרו DialogBox משתמש בפונקציה CreateWindowEx כדי ליצור את תיבת הדו-שיח, ואחר כך היא שולחת למאקרו תיבת הדו-שיח את ההודעה WM_INITDIALOG (והודעה WM_SETFONT, בתנאי שהסגנון DS_SETFONT הוגדר בתבנית). הפונקציה מציגה את תיבת הדו-שיח (ללא תלות בהגדרה של הסגנון WS_VISIBLE בתבנית), מעבירה את חלון האב למצב לא פעיל, ומתחילה בלולאת הודעות משלה לקבלת הודעות וביצוען עבור תיבת הדו-שיח.

כאשר פונקציית הטיפול בהודעות של תיבת הדו-שיח קוראת לפונקציה EndDialog, DialogBox הורסת את תיבת הדו-שיח, מסיימת את לולאת הודעות, מחזירה את חלון האב למצב פעיל (אם הועבר למצב לא פעיל קודם לכן), ומחזירה לחלון שקרא לה את הפרמטר nResult.

5.7 לולאת ההודעות של תיבת הדו-שיח

כשיוצרים תיבות דו-שיח, תיבת הדו-שיח משתמשת בפונקציית הודעות משלה, ולא בפונקציית ההודעות של WndProc שחלון האב משתמש בה. ייתכן, שבתוכניות שתכתוב יהיו פונקציות הודעות שונות, אפילו כמספר תיבות הדו-שיח שיש בתוכנית, פונקציה אחת עבור כל תיבת דו-שיח. באפשרותך לתת כל שם שתבחר לפונקציית ההודעות, מכיון שאתה מוסר את כתובת הפונקציה בכל פעם שאתה יוצר תיבת דו-שיח. לדוגמה, בסעיף 5.9 בתוכנית DlgBox תיבת הדו-שיח נוצרת על ידי הקריאה הבאה של הפונקציה DialogBox :

```
DialogBox(hInst, "TestDialog", hWnd, (DLGPROC)TestDlgProc);
```

הפרמטר האחרון, המצביע אל פונקציית ההודעות, מורה ל-Windows לאן צריך לשלוח את ההודעות שמיועדות לתיבת הדו-שיח. כפי שניתן לראות בקטע הקוד הבא, פונקציית ההודעות של תיבת הדו-שיח מבצעת טיפול דומה לזה שמבצעת הפונקציה WndProc :

```
LRESULT CALLBACK TestDlgProc(HWND hDlg, UINT uMsg,
                             WPARAM wParam, LPARAM lParam)
{
  switch( uMsg )
  {
    case WM_INITDIALOG :
      break;
    case WM_COMMAND :
       switch( LOWORD( wParam ) )
        {
          case IDOK :
             EndDialog( hDlg, IDOK );
             break;
```

```
          case IDCANCEL :
              EndDialog( hDlg, IDCANCEL );
              break;
          }
        break;
   default :
        return( FALSE );
  }
  return( TRUE );
}
```

במקרה של פונקציית ההודעות של תיבת הדו-שיח שקטע הקוד מפרט, הפונקציה בודקת שתי הודעות בסיסיות: WM_INITDIALOG ו-WM_COMMAND. כאשר הפונקציה מקבלת את ההודעה WM_INITDIALOG, היא לא עושה כרגע כלום. כאשר הפונקציה מקבלת את ההודעה WM_COMMAND, היא בודקת את הערך של מילת הסדר-הנמוך של הפרמטר wParam. כידוע לך, מילת הסדר-הנמוך של הפרמטר wParam מכילה את הקבוע של הפקודה שנבחרה על ידי המשתמש. כשהמשתמש לוחץ בעכבר על OK (ו-wParam מכיל את הקבוע IDOK), הפונקציה מחזירה IDOK. עם זאת, כשהמשתמש לוחץ בעכבר על Cancel (ו-wParam מכיל את הקבוע IDCANCEL), הפונקציה סוגרת את תיבת הדו-שיח מבלי לשמור את השינויים שהמשתמש עשוי היה לעשות בתיבה.

5.8 השימוש במקלדת עם תיבות דו-שיח

כבר ראית ש-Windows תומכת במנגנון לוגי מיוחד לטיפול בתיבות דו-שיח, המתבטא בפונקציית תיבת דו-שיח לטיפול בהודעות. תמיכה זו כוללת גם אמצעים לבחירת פריטים בתיבת הדו-שיח על ידי הקשות מקשים כתחליף ללחיצות בעכבר. אפשר לחלק את הלוגיקה שבתוכנית לשימוש במקלדת לשלוש קבוצות: **מקשים מהירים** (מקשים חמים, Hot Keys), שבעזרתם בוחרים פריטים בחלון בשיטת "Alt+(אות)" כתגובה לפעולת הטבלר (Tab), אשר מעביר את המיקוד בין הפקדים, או כתגובה למקשי החיצים שגם הם מעבירים מיקוד בין הפקדים ובתוכם.

הענקת תמיכה בתיבות הדו-שיח לצירוף "Alt+(אות)" של המקלדת נעשה באותה דרך שבה השתמשת עבור פריטי תפריט. בתוך מחרוזות הטקסט של הפקד צריך להכניס את התו "&" **לפני** האות שרוצים להשתמש בה בצירוף ההקשה. לדוגמה, ההגדרה הבאה מיועדת עבור הפקד DEFPUSHBUTTON, והיא משתמשת בצירוף ההקשות Alt+D כמקש קיצור להפעלת הלחצן "Dalmatian":

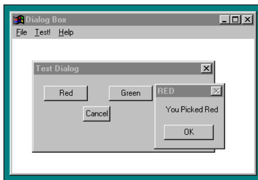
```
CONTROL "&Dalmatian", IDC_DONE, "BUTTON",
   BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD, 45, 66, 48, 12
```

לפעמים המשתמשים ימצאו שהשימוש בפקדי מקלדת נוח יותר מאשר לחצן העכבר. אולם כמו כל שאר האמצעים האחרים שעומדים לרשות המשתמש להפעלת התוכנית בקלות, צריך להיזהר ולא להגדיר יותר מדי מקשים מהירים, מכיוון שהדבר רק עלול לבלבל את המשתמש, ולא לעזור לו. כדי לתמוך בפקדי מקלדת בתיבות דו-שיח, חייבים לקבוע מספר אלמנטים בתבנית תיבת הדו-שיח, ובכלל זה הסגנונות WS_GROUP-ו WS_TABSTOP.

הסגנון WS_TABSTOP מסמן כל פריט שמקבל את **מיקוד הקלט** כאשר המשתמש מקיש על Tab או Shift+Tab. הסגנון WS_GROUP מסמן התחלת קבוצה. כל הפריטים הרשומים **בשפת תסריט** בקובץ המשאבים עד להגדרת הסגנון הבא של WS_GROUP, הם חלק מקבוצה אחת. המשתמש יכול להשתמש במקשי החיצים כדי לעבור בין הפריטים בתוך הקבוצה, אך הוא אינו יכול להשתמש במקשי החיצים כדי לעבור מקבוצה אחת לקבוצה אחרת.

5.9 תוכנית ראשונה ליצירת תיבת דו-שיח

נציג את התוכנית השלמה ליצירת תיבת דו-שיח. כאשר התוכנית **DlgBox** הנמצאת בתקליטור (Chap05) מתחילה לפעול, מוצג בשורת התפריט רק התפריט ברמה העליונה. על ידי בחירה באפשרות !Test, המשתמש גורם להצגת תיבת הדו-שיח. מרגע שתיבת הדו-שיח מוצגת על המסך, בחירה באחד הלחצנים תגרום לתגובה מתאימה. מסך לדוגמה נראה בתרשים 5.2.



תרשים 5.2: פלט לדוגמה של התוכנית הראשונה ליצירת תיבות דו-שיח

```
#include <windows.h>
#include "DlgBox.h"

#if defined (WIN32)
                        #define IS_WIN32 TRUE
#else
                        #define IS_WIN32 FALSE
#endif

#define IS_NT       IS_WIN32 && (BOOL)(GetVersion() < 0x80000000)
#define IS_WIN32S   IS_WIN32 && (BOOL)(!(IS_NT) &&
                    (LOBYTE(LOWORD(GetVersion()))<4))
#define IS_WIN95    (BOOL)(!(IS_NT) && !(IS_WIN32S)) && IS_WIN32

HINSTANCE hInst;   // current instance

LPCTSTR lpszAppName  = "DlgBox";
LPCTSTR lpszTitle    = "Dialog Box";

BOOL RegisterWin95( CONST WNDCLASS* lpwc );

int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                  hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
{
   MSG       msg;
   HWND      hWnd;
   WNDCLASS wc;

   // Register the main application window class.
   //...........................................
   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = (WNDPROC)WndProc;
   wc.cbClsExtra     = 0;
   wc.cbWndExtra     = 0;
   wc.hInstance      = hInstance;
   wc.hIcon          = LoadIcon( hInstance, lpszAppName );
   wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName;
```

```
   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );

   hInst = hInstance;

   // Create the main application window.
   //...................................
   hWnd = CreateWindow( lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                      );

   if ( !hWnd )
      return( FALSE );

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );

   while( GetMessage( &msg, NULL, 0, 0) )
   {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }

   return( msg.wParam );
}
```

```
BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style         = lpwc->style;
   wcex.lpfnWndProc   = lpwc->lpfnWndProc;
   wcex.cbClsExtra    = lpwc->cbClsExtra;
   wcex.cbWndExtra    = lpwc->cbWndExtra;
   wcex.hInstance     = lpwc->hInstance;
   wcex.hIcon         = lpwc->hIcon;
   wcex.hCursor       = lpwc->hCursor;
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;

   // Added elements for Windows 95.
   //..............................
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");

   return RegisterClassEx( &wcex );
}

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        DialogBox( hInst, "TestDialog", hWnd,
                                   (DLGPROC)TestDlgProc );
                        break;
                 case IDM_EXIT :
                        DestroyWindow( hWnd );
                        break;
              }
              break;
```

```
case WM_DESTROY :
              PostQuitMessage(0);
              break;

      default :
           return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
   }

   return( 0L );
}

LRESULT CALLBACK TestDlgProc( HWND hDlg, UINT uMsg,
                              WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {

      case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
               case IDOK :
                    EndDialog( hDlg, IDOK );
                    break;
               case IDCANCEL :
                    EndDialog( hDlg, IDCANCEL );
                    break;
               case IDD_RED:
                    MessageBox(hDlg, "You Picked Red", "RED",
                               MB_OK);
                    break;
               case IDD_GREEN:
                    MessageBox(hDlg, "You Picked Green", "GREEN",
                               MB_OK);
                    break;
                       }
            break;

      default :
            return( FALSE );
   }

   return( TRUE );
}
```

```
LRESULT CALLBACK About( HWND hDlg,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
   switch (message)
   {
        case WM_INITDIALOG:
                return (TRUE);

       case WM_COMMAND:
       switch(LOWORD(wParam)) {
         case IDCANCEL:
           EndDialog(hDlg, 0);
           return 1;

           return 1;
       }
   }

   return (FALSE);
}
```

שים לב למשתנה הגלובלי hInst. משתנה זה מקבל עותק של ידית המופע הנוכחי המועברת לפונקציה ()WinMain. הסיבה לקיומו של משתנה זה היא, שתיבת הדו-שיח זקוקה לגישה לידית של המופע הנוכחי. תיבת הדו-שיח לא נוצרת במסגרת הפונקציה ()WinMain, אלא בפונקציה ()WndProc, ולכן יש ליצור עותק של פרמטר המופע, כדי שניתן יהיה לגשת אליו גם מחוץ ל-()WinMain.

5.10 אתחול נתונים בתיבת דו-שיח

בסעיף 5.7 למדת על פונקציית ההודעות של תיבת הדו-שיח, שמטפלת בהודעות ש-Windows שולחת לתיבת הדו-שיח. במקרה של התוכנית **dlgbox**, פונקציית ההודעות מאתחלת את הפקדים (בערך שנמצא במשתנה הגלובלי), ולפי הקלט שהיא מקבלת מהמשתמש, שומרת את הערך של הפקד, במשתנה הגלובלי המתאים. פונקציית ההודעות משתמשת בפונקציות SetDlgItemInt ו-SetDlgItemText כדי לאתחל את הפקדים של תיבת הדו-שיח, ובפונקציות GetDlgItemInt ו-GetDlgItemText - כדי לקבל את ערכי הפקדים. פעמים רבות תשתמש בפונקציות אלו בתוכניות, כדי לאתחל את הפקדים ולקבל את הערכים שהם מכילים. להלן ההגדרות של פונקציות אלו.

```
BOOL SetDlgItemInt(HWND hDlg, int nIDDlgItem,
                        UINT uValue, BOOL bSigned);
BOOL SetDlgItemText(HWND hDlg, int nIDDlgItem,
                        LPCTSTR lpString);
UINT GetDlgItemInt(HWND hDlg, int nIDDlgItem,
                        BOOL *lpTranslated, BOOL bSigned);
UINT GetDlgItemText(HWND hDlg, int nIDDlgItem,
                        LPCTSTR lpString, int nMaxCount);
```

שים לב ששתי פונקציות Set מחזירות ערך BOOL, שמציין את הצלחת הפונקציה או כישלונה; וכך, שתי פונקציות Get מחזירות ערך UINT (Unsigned Integer). עבור הפונקציה GetDlgItemText, הערך המוחזר מציין את מספר התווים שהפונקציה העתיקה למאגר lpString. טבלה 5.4 מפרטת את הפרמטרים עבור ארבעת הפונקציות ומראה את הפרמטר המתאים לכל אחת מהן.

טבלה 5.4: הפרמטרים של הפונקציות SetDlgItemInt, SetDlgItemText, GetDlgItemInt, ו-GetDlgItemText.

פרמטר	פונקציות	תיאור
hDlg	SetDlgItemInt SetDlgItemText GetDlgItemInt GetDlgItemText	מציין את תיבת הדו-שיח שמכילה את הפקד.
nIDDlgItem	SetDlgItemInt SetDlgItemText GetDlgItemInt GetDlgItemText	מגדיר איזה פקד לשנות. מגדיר את הפקד שצריך לקבל ממנו את הערך.
uValue	SetDlgItemInt	מגדיר את הערך של מספר שלם אשר מתייחסים לספרותיו כמחרוזת טקסט שתוצג בפקד (בקיצור, מציג את המספר שיוצג בפקד).
bSigned	SetDlgItemInt GetDlgItemInt	מגדיר אם הפרמטר uValue **מסומן** (Signed) או **לא מסומן** (Unsigned). עבור GetDlgItemInt, הוא מגדיר אם הערך המוחזר מסומן או לא מסומן. אם פרמטר זה הוא True, אז uValue מסומן. אם פרמטר זה הוא True והערך של uValue קטן מאפס, SetDlgItemInt מציבה את הסימן מינוס לפני הספרה הראשונה של המספר. אם פרמטר זה הוא False, uValue לא מסומן.

פרמטר	פונקציות	תיאור
lpString (אות ראשונה - L)	SetDlgItemText GetDlgItemText	עבור SetDlgItemText, ‏lpString מגדיר את המחרוזת שצריך להציג בפקד. עבור GetDlgItemText, הוא מגדיר מאגר עבור המחרוזת שבה הפונקציה מציבה את הערך המוחזר מהפקד.
lpTranslated (אות ראשונה - L)	GetDlgItemInt	מצביע למשתנה בוליאני שמקבל את הערך שמציין הצלחה (True), או כישלון (False) של הפונקציה. פרמטר זה הוא רשות, ויכול להיות NULL. כאשר ערכו NULL, הפונקציה אינה מחזירה מידע על הצלחה או על כישלון.
nMaxCount	GetDlgItemText	הפונקציה מגדירה את הגבול של מספר התווים שתקרא מהפקד למאגר המחרוזת שמוצבע על ידי lpString.

5.11 המאקרו CreateDialog

בסעיף 5.6 השתמשת במאקרו DialogBox כדי ליצור תיבת דו-שיח מודאלית ממשאב של תבנית תיבת דו-שיח. גם למדת שלפעמים צריך ליצור תיבות דו-שיח לא מודאליות, בנוסף לתיבות המודאליות. באפשרותך להשתמש במאקרו CreateDialog כדי ליצור תיבות דו-שיח לא מודאליות. המאקרו CreateDialog יוצר תיבות דו-שיח לא מודאליות ממשאב של תבנית תיבת דו-שיח. המאקרו CreateDialog משתמש בפונקציה CreateDialogParam, שמוסברת בפירוט בסעיף 5.12. את המאקרו CreateDialog כותבים על פי ההגדרה שלהלן:

```
HWND CreateDialog(
   HINSTANCE, hInstance, // handle to application instance
   LPCTSTR lpTemplate,   // identifies dialog box template name
   HWND hWndParent,      // handle to owner window
   DLGPROC lpDialogFunc  // pointer to dialog box procedure
);
```

ניתן לראות, שהמאקרו CreateDialog מקבל ארבעה פרמטרים.

המאקרו CreateDialog משתמש בפונקציה CreateWindowEx כדי ליצור את תיבת הדו-שיח. אחר כך, CreateDialog שולחת את ההודעה WM_INITDIALOG (וגם את WM_SETFONT, אם הוגדר בתבנית הסגנון DS_SETFONT) אל מאקרו תיבת הדו-שיח. המאקרו מציג את תיבת הדו-שיח אם הוגדר בתבנית הסגנון WM_VISIBLE, ולבסוף - CreateDialog מחזיר את ידית החלון של תיבת הדו-שיח.

לאחר חזרה מהמאקרו CreateDialog, היישום משתמש בפונקציה ShowWindow להצגת תיבת הדו-שיח (אם עדיין לא הוצגה). היישום משתמש בפונקציה DestroyWindow כדי להרוס את תיבת הדו-שיח. כדי להבין יותר טוב את הטיפול שמבצע CreateDialog, התבונן בתוכנית **Create_Dialog** שבתקליטור המצורף לספר זה (בתיקיה chap05). כאשר המשתמש בוחר באפשרות !Test, הפונקציה WndProc יוצרת תיבת דו-שיח לא מודאלית פשוטה, כמו בקטע הקוד שלהלן:

```
case IDM_TEST :
   if (!hDlgModeless)
     hDlgModeless = CreateDialog(hInst, "TestDialog", hWnd,
                                  (DLGPROC)TestDlgProc );
   break;
```

כאשר המשתמש בוחר באפשרות !Test, התוכנית **Create_Dialog** בודקת כדי לקבוע אם היא מציגה כרגע את תיבת הדו-שיח הלא מודאלית. אם כן, היא אינה מבצעת עיבוד; אם לא, **Create_Dialog** משתמשת בפונקציה CreateDialog כדי להציג את תיבת הדו-שיח הלא מודאלית.

5.12 הפונקציה CreateDialogParam

בסעיף 5.6 למדת להשתמש במאקרו DialogBox כדי ליצור תיבות דו-שיח מודאליות, ובסעיף 5.11 למדת להשתמש במאקרו CreateDialog כדי ליצור תיבות דו-שיח לא מודאליות. כאשר חייבים ליצור תיבות דו-שיח לא מודאליות, אפשר להשתמש גם בפונקציה CreateDialogParam. פונקציה זו (שהמאקרו CreateDialog קורא לה כחלק מהעיבוד שהיא מבצעת) יוצרת תיבת דו-שיח לא מודאליות ממשאב תבנית של תיבת דו-שיח. הפונקציה CreateDialogParam מאפשרת גם להעביר ערך שמוגדר על ידי היישום (Application-Defined Value), אל מאקרו תיבת הדו-שיח בפרמטר lParam של ההודעה WM_INITDIALOG. באפשרות היישום להשתמש בערך הפרמטר lParam כדי לאתחל את פקדי תיבת הדו-שיח. משתמשים בפונקציה CreateDialogParam בתוכניות, כמו בדוגמה הכללית הזו:

```
HWND CreateDialogParam(
   HINSTANCE, hInstance,    // handle to application instance
   LPCTSTR lpTemplateName,  // identifies dialog box template
   HWND hWndParent,         // handle to owner window
   DLGPROC lpDialogFunc     // pointer to dialog box procedure
   LPARAM dwInitParam       // initialization value
);
```

הפונקציה CreateDialogParam מקבלת חמישה פרמטרים. הפרמטר dwInitParam מגדיר את הערך שצריך למסור לפונקציית תיבת הדו-שיח בפרמטר lParam של ההודעה WM_INITDIALOG.

הפונקציה CreateDialogParam משתמשת בפונקציה CreateWindowEx כדי ליצור את תיבת הדו-שיח. אחר כך, CreateDialogParam שולחת אל פונקציית תיבת הדו-שיח את ההודעה WM_INITDIALOG (וגם את WM_SETFONT אם הוגדר בתבנית הסגנון DS_SETFONT). הפונקציה מציגה את תיבת הדו-שיח אם הוגדרה בתבנית הסגנון WM_VISIBLE. לבסוף, CreateDialog מחזירה את ידית החלון של תיבת הדו-שיח אם הצליחה ליצור את תיבת הדו-שיח.

אחרי ש-CreateDialogParam חוזרת, היישום משתמש בפונקציה ShowWindow להצגת תיבת הדו-שיח (אם עדיין לא הוצגה). היישום משתמש בפונקציה DestroyWindow כדי לפרק את תיבת הדו-שיח. כדי להבין יותר טוב את מהלך העיבוד שמבצעת הפונקציה CreateDialogParam, התבונן בתוכנית **Create_Dialog**, שנמצאת בתקליטור המצורף לספר (בתיקיה Books\59285). כאשר המשתמש בוחר באפשרות !Test, הפונקציה WndProc יוצרת תיבת דו-שיח לא מודאלית פשוטה, כמו שרואים בקטע הקוד שלהלן:

```
case IDM_TEST :
  if ( !hDlgModeless )
   {
     hDlgModeless = CreateDialogParam(hInst, "TestDialog",
               hWnd, (DLGPROC)TestDlgProc, (LPARAM)lpMem );
   }
  break;
```

התוכנית **Create_Dialog** מגדירה מבנה (Structure) מסוג DLGDATA, כדי להחזיק בו את המצב הנוכחי עבור כל לחצן שבתיבת הדו-שיח, כמו בדוגמה זו:

```
typedef struct {
   BOOL bChecked;
   BOOL bRadio1;
} DLGDATA;
```

כאשר מתבצע אתחול לתוכנית (כלומר, כאשר הפונקציה WndProc מקבלת את ההודעה WM_CREATE), התוכנית מקצה ידית למופע (Instance) של המבנה. כאשר המשתמש בוחר באפשרות !Test, התוכנית מעבירה ידית זו כפרמטר אחרון בפונקציה CreateDialogParam. הפונקציה TestDlgProc מקבלת את הידית בפרמטר lParam ומשתמשת בערכים, כדי לאתחל את מצב הפקדים בתיבת הדו-שיח.

5.13 ברירת המחדל לטיפול בהודעות של תיבת דו-שיח

למדת שחובה ליצור בתוכנית פונקציית טיפול בהודעות תיבות הדו-שיח כדי לטפל, ולעבד, את ההודעות ש-Windows שולחת לתיבות הדו-שיח. למדת שפונקציית תיבות הדו-שיח לטיפול בהודעות מבצעת צעדים דומים לפונקציה WndProc, שמטפלת בהודעות החלון שבתוכנית שלך. כשלמדת על הפונקציה WndProc, ראית שהתוכניות צריכות להגדיר תמיד את פונקציית ברירת המחדל של Windows לטיפול בהודעות (DefWindowProc), במשפט Case האחרון של הוראת Switch, כפי שמוצג להלן:

```
default :
      return(DefWindowProc(hWnd, uMsg, wParam, lParam));
   }
```

אם אתה מגדיר **מחלקת חלון** (Window Class) נפרדת כדי ליצור את חלון תיבת הדו-שיח, עליך גם להגדיר פונקציית ברירת מחדל לטיפול בהודעות של תיבות הדו-שיח, כמו שאתה נוהג בחלון הבסיסי. כדי להגדיר פונקציית ברירת מחדל לטיפול בהודעות עבור תיבות הדו-שיח, עליך להשתמש בתוכנית שלך בפונקציה DefDlgProc. הפונקציה DefDlgProc מבצעת את טיפול ברירת המחדל בהודעות עבור פונקציית חלון ששייכת למחלקת תיבת דו-שיח שהוגדרה על ידי היישום (application-defined dialog box class). משתמשים בפונקציה DefDlgProc בתוכניות, כפי שמוצג בדוגמה הכללית שלהלן:

```
LRESULT DefDlgProc(
   HWND hDlg,            // handle to dialog box
   UINT, Msg,            // message
   WPARAM wParam,        // first message parameter
   LPARAM lParam         // second message parameter
);
```

הפונקציה DefDlgProc קובעת את ברירת המחדל של מחלקת החלון שהוגדרה עבור תיבת הדו-שיח. פונקציה זו מספקת טיפול פנימי לתיבת הדו-שיח על ידי משלוח ההודעות לפונקציית תיבת הדו-שיח וטיפול ברירת מחדל לכל הודעה שפונקציית תיבת הדו-שיח מחזירה כ-False. יישומים שיוצרים פונקציות חלון מותאמות אישית עבור תיבות הדו-שיח המותאמות אישית, משתמשים לפעמים בפונקציה DefDlgProc במקום בפונקציה DefWindowProc לצורך טיפול ברירת המחדל בהודעות.

יוצרים ביישומים מחלקות תיבות דו-שיח מותאמות אישית, על ידי הצבת מידע מתאים במבנה WNDCLASS ועל ידי רישום המחלקה על ידי הפונקציה RegisterClass. חלק מהיישומים משתמשים בפונקציה GetClassInfo להצבת הערכים במבנה ולקביעת שם תיבת הדו-שיח שהוגדרה.

במקרים כאלה, היישום מגדיר לפחות את האיבר lpszClassName לפני שהוא רושם את המחלקה. בכל המקרים, אתה חייב לקבוע את ערך האיבר cbWndExtra של המבנה WNDCLASS לפחות כ-DLGWINDOWEXTRA, עבור תיבות דו-שיח מותאמות אישית.

אסור לפונקציית תיבת דו-שיח לקרוא לפונקציה DefDlgProc; כי הדבר יגרום ל**הפעלה רקורסיבית** (Recursive Execution). כדי להבין יותר טוב את הפעולה של הפונקציה DefDlgProc, התבונן בתוכנית **DefDlgP**, שנמצאת בתקליטור המצורף לספר זה (בתיקיה Chap05\59285).

כאשר מתבוננים בהגדרת הפונקציה WndProc, צריך לשים לב שההודעה WM_CREATE במשפט Switch מאתחלת את **מחלקת תיבת הדו-שיח** (Dialog Box Class). הגדרת המשאב עושה את תיבת הדו-שיח למחלקה נפרדת, כפי שמוצג להלן:

```
TESTDIALOG DIALOG DISCARDABLE  0, 0, 180, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
                    WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
CLASS "BlueDlg"
BEGIN
  CHECKBOX      "Check box control.", IDC_CHECKBOX,9,7,70,10
  GROUPBOX      "Radio Buttons",-1,7,21,86,39
  RADIOBUTTON       "First",IDC_RADIO1,13,32,37,10,WS_GROUP |
                     WS_TABSTOP
  RADIOBUTTON       "Second",IDC_RADIO2,13,45,39,10
  PUSHBUTTON        "Done",IDCANCEL,116,8,50,14,WS_GROUP
END
```

כשמפעילים את התוכנית **DefDlgP** ובוחרים באפשרות !Test, התוכנית תשתמש במחלקה BlueDlg כדי ליצור את תיבת הדו-שיח. עבור כל הודעת Windows שתיבת הדו-שיח אינה מטפלת בה, DefDlgProc קוראת לפונקציית ברירת המחדל לטיפול בתיבת הדו-שיח (במקרה של התוכנית **DefDlgP**, פונקציית ברירת המחדל היא TestDlgProc).

5.14 סגירת תיבת הדו-שיח - EndDialog

בסעיפים קודמים למדת ליצור ולהציג סוגים שונים של תיבות דו-שיח. ראית בקוד שהוצג בסעיפים אלה שחייבים להשתמש בפונקציה EndDialog לסגירת תיבת הדו-שיח המודאלית. הפונקציה EndDialog מפרקת את תיבת הדו-שיח המודאלית, והדבר גורם למערכת לסיים כל טיפול בה. משתמשים בפונקציה EndDialog כפי שמוצג להלן:

```
BOOK EndDialog(
  HWND hDlg,          // handle to dialog box
  int nResult,        // value to return
);
```

הפרמטר hDlg הוא שם תיבת הדו-שיח שהפונקציה EndDialog צריכה לפרק. הפרמטר nResult מאפשר לתוכנית להגדיר ערך מוחזר מהפונקציה שיצרה את תיבת הדו-שיח אל היישום הקורא. אתה חייב להשתמש בפונקציה EndDialog כדי לפרק תיבות דו-שיח שיצרת על ידי הפונקציות DialogBox, DialogBoxParam, DialogBoxIndirect או DialogBoxIndirectParam. היישום קורא לפונקציה EndDialog מתוך פונקציית תיבת דו-שיח. אל תשתמש בפונקציה EndDialog לכל מטרה אחרת.

פונקציית תיבת דו-שיח יכולה לקרוא לפונקציה EndDialog בכל זמן, אפילו בזמן הטיפול בהודעה WM_INITDIALOG. אם היישום קורא לפונקציה EndDialog בזמן הטיפול בהודעה WM_INITDIALOG, Windows מפרקת את תיבת הדו-שיח לפני שהיא מציגה אותה, ולפני שהיא מעבירה את מיקוד הקלט (Input Focus) לתיבת הדו-שיח.

EndDialog אינה מפרקת את תיבת הדו-שיח מייד. במקום זה, היא קובעת דגל ומאפשרת לפונקציית תיבת הדו-שיח להחזיר את השליטה למערכת. המערכת בודקת את הדגל לפני שהיא מנסה לקבל את ההודעה הבאה מהתור של היישום. אם הפונקציה EndDialog קבעה כבר את הדגל, המערכת מסיימת את לולאת ההודעות, מפרקת את תיבת הדו-שיח ומשתמשת בערך שנמצא בפרמטר nResult כערך מוחזר מהפונקציה שיצרה את תיבת הדו-שיח.

5.15 כיצד להוסיף תיבת רשימה

נמשיך ונבחן את תיבות הדו-שיח על ידי הוספת פקד לתיבת הדו-שיח שהגדרנו בתוכנית הקודמת. אחד הפקדים הנפוצים ביותר, אחרי הלחצנים, הוא תיבות רשימה (List Box). להלן ההגדרה הכללית של משפט LISTBOX:

```
LISTBOX LBID, X, Y, Width, Heigh [,Style]
```

במקרה זה, LBID הוא הערך המגדיר את תיבת הרשימה. הפינה השמאלית-העליונה של התיבה תמוקם בנקודה X,Y, וגודל התיבה יהיה על פי ההגדרות של הפרמטר Width והפרמטר Height. הפרמטר Style קובע את הסגנון המדויק של התיבה (ערכי Style המשמשים אותנו פה הם אותם ערכים המופיעים בטבלה 5.1).

כדי להוסיף תיבת רשימה, עליך לשנות את ההגדרה של תיבת הדו-שיח בקובץ המשאבים ListBox.RC (הקבצים לפני השינויים מופיעים בתיקיה: Chap05\ListBox\Parts). הוסף תחילה להגדרת תיבת הדו-שיח את תיאור תיבת הרשימה המובא להלן:

```
LISTBOX ID_LB1, 75,28,96,39, LBS_NOTIFY ¦ WS_CHILD ¦
        WS_VISIBLE ¦ WS_BORDER ¦ WS_VSCROLL ¦ WS_TABSTOP
```

עתה, הוסף להגדרת תיבת הדו-שיח את הלחצן הבא:

```
PUSHBUTTON      "Select Fruit",IDD_SELFRUIT,10,29,38,14
```

לאחר שתכניס שינויים אלה, תיראה ההגדרה של תיבת הדו-שיח כך:

```
TESTDIALOG DIALOG DISCARDABLE  20, 20, 178, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON    "Red",IDD_RED,10,10,44,14
    PUSHBUTTON       "Green",IDD_GREEN,75,10,44,14
    PUSHBUTTON       "Cancel",IDCANCEL,48,29,27,14
    PUSHBUTTON       "Select Fruit",IDD_SELFRUIT,10,29,38,14
    LISTBOX          ID_LB1,75,28,96,39,WS_VSCROLL | WS_TABSTOP
END
```

לבסוף, עליך להוסיף לקובץ ListBox.h גם את פקודות המאקרו האלו:

```
#define ID_LB1          1042
#define IDD_SELFRUIT    1043
```

ID_LB1 מגדיר את תיבת הרשימה שצוינה בהגדרת תיבת הדו-שיח שבתוך קובץ המשאבים. IDD_SELFRUIT הוא ערך ID של הלחצן Select Fruit.

עקרונות בסיסיים בהפעלת תיבות רשימה

כשאתה משתמש בתיבת רשימה, עליך לבצע שתי פעולות בסיסיות. ראשית, עליך לאתחל את תיבת הרשימה כאשר תיבת הדו-שיח מוצגת לראשונה. כלומר, לשלוח לתיבה את הרשימה שתוצג בה (לפי ברירת המחדל, תיבת הרשימה תוצג כשהיא ריקה). שנית, לאחר שתיבת הרשימה אותחלה, תצטרך התוכנית שלך להגיב בכל פעם שהמשתמש יבחר בפריט כלשהו מתוך הרשימה.

תיבות רשימה מפיקות סוגים שונים של הודעות. הסוג היחיד שישמש אותנו כאן הוא ההודעה LBN_DBLCLK. הודעה כזו נשלחת כאשר המשתמש לוחץ בעכבר לחיצה כפולה על אחד הפריטים ברשימה. הודעה זו מוכנסת לתוך HIWORD(wParam) כל פעם שמופקת עבור תיבת הרשימה הודעת WM_COMMAND (חובה לכלול בהגדרת תיבת הרשימה דגל בסגנון LBS_NOTIFY, כדי שתוכל להפיק הודעות מסוג LBN_DBLCLK). לאחר שנבחרה אפשרות מסוימת, עליך לבחון את תיבת הרשימה כדי לגלות איזה פריט נבחר.

שלא כמו הלחצן, תיבת רשימה היא אמצעי בקרה המקבל ומפיק הודעות כאחד. תוכל לשלוח לתיבת רשימה מספר הודעות שונות, אולם בדוגמה שלנו, נשלח רק את שתי ההודעות שלהלן.

שם המאקרו	הפעולה
LB_ADDSTRING	מוסיף מחרוזת (בחירה) לתיבת הרשימה
LB_GETCURSEL	מבקש את האינדקס של הפריט שנבחר

LB_ADDSTRING היא הודעה המורה לתיבת הרשימה להוסיף לרשימה מחרוזת מסוימת. כלומר, המחרוזת שצוינה הופכת לפריט נוסף בתיבה. מייד תראה כיצד להשתמש בהודעה זו. ההודעה LB_GETCURSEL גורמת לתיבת הרשימה להחזיר את האינדקס של הפריט שהמשתמש בחר מתוך הרשימה. כל מספרי האינדקס של תיבות רשימה מתחילים בסיפרה 0 (על כן, הפריט השלישי ברשימה יקבל אינדקס 2).

כדי לשלוח הודעה לתיבת הרשימה (או לכל פקד אחר), השתמש בפונקציית API בשם SendDlgItemMessage(), שהגדרתה היא:

```
LONG SendDlgItemMessage(HWND hdwnd, int ID,UINT IDMsg,
                        WPARAM wParam, LPARAM lParam);
```

הפונקציה SendDlgItemMessage() שולחת את ההודעה שמכיל הפרמטר IDMsg לאותו פקד (בתיבת הדו-שיח), אשר מספר ID שלו מופיע בפרמטר ID. הידית לתיבת הדו-שיח נמצאת ב-hdwnd. כל מידע נוסף הדרוש להודעה כלול בפרמטרים wParam ו- lParam. המידע הנוסף, אם ישנו, משתנה מהודעה להודעה. אם אין מידע נוסף להעברה לפקד, ערכי הפרמטרים wParam ו-lParam יהיו שניהם אפס (0). הערך שמחזירה הפונקציה SendDlgItemMessage() מכיל את המידע המיועד ל-IDMsg.

כיצד לאתחל את תיבת הרשימה

הואיל ולפי ברירת המחדל תיבת רשימה מופיעה כשהיא ריקה, עליך לאתחל אותה כל פעם שמוצגת תיבת הדו-שיח שאליה היא משתייכת. פעולה זו פשוטה למדי, מכיון שבכל פעם שמופעלת תיבת דו-שיח, נשלחת לפונקציית החלון שלה הודעת WM_INITDIALOG. לכן, עליך להוסיף את משפט case שלהלן, למשפט switch החיצוני שבפונקציה TestDlgProc () (Switch(Message)).

```
case WM_INITDIALOG: // initialize list box
      SendDlgItemMessage(hDlg, ID_LB1,
                          LB_ADDSTRING, 0, (LPARAM)"Apple");
      SendDlgItemMessage(hDlg, ID_LB1,
                          LB_ADDSTRING, 0, (LPARAM)"Orange");
      SendDlgItemMessage(hDlg, ID_LB1,
                          LB_ADDSTRING, 0, (LPARAM)"Pear");
      SendDlgItemMessage(hDlg, ID_LB1,
                          LB_ADDSTRING, 0, (LPARAM)"Grape");
```

קטע זה גורם לטעינת תיבת הרשימה במחרוזות "Apple", "Orange", "Pear" ו- "Grape". הוספת כל אחת מהמחרוזות לתיבת הרשימה נעשית על ידי קריאה לפונקציה ()SendDlgItemMessage, אשר נעשית בעזרת הודעת LB_ADDSTRING. הפרמטר lParam מצביע בכל מקרה על המחרוזת המיועדת להוספה (חובה להעביר את הסוג אל LPARAM). במקרה זה, המחרוזות מתווספות לתיבת הרשימה בסדר שבו נשלחו, אולם בהתאם לצורה שבה בנית את תיבת הרשימה, ניתן לגרום להצגת הפריטים בסדר האלף-בית. אם מספר הפריטים ששלחת לתיבת רשימה עולה על מה שניתן להציג בחלון התיבה, יתוספו לה באופן אוטומטי פסי גלילה אנכיים.

כיצד לעבד הודעה על בחירת פריט

לאחר שאותחלה, תיבת הרשימה מוכנה לשימוש. בעיקרון, קיימות שתי דרכים לבחירת אפשרות מתיבת רשימה. ראשית, המשתמש יכול ללחוץ בעכבר לחיצה כפולה על אחד הפריטים בתיבה. דבר זה גורם להעברת הודעת WM_COMMAND לפונקציית החלון של תיבת הדו-שיח. במקרה זה, מכיל הפרמטר (LOWORD(wParam את ה-ID הקשור בתיבת הרשימה, ואילו הפרמטר (HIWORD(wParam מכיל את ההודעה LBN_DBLCLK. לחיצה כפולה מודיעה לתוכנית מייד על הבחירה של המשתמש. הדרך השנייה להשתמש בתיבת רשימה היא לסמן את אחד הפריטים. דבר זה **אינו** גורם למשלוח הודעה לתוכנית, אך תיבת הרשימה זוכרת את הבחירה, ומחכה עד שהתוכנית תבקש לדעת מהי. שתי השיטות מודגמות בתוכנית המובאת לפניך.

לאחר שפריט מסוים נבחר מתוך תיבת רשימה, עליך לקבוע מיהו הפריט שנבחר על ידי שיגור הודעת LB_GETCURSEL לתיבת הרשימה. התיבה תחזיר את מספר האינדקס של הפריט שנבחר. אם ההודעה נשלחת לפני שנבחר פריט כלשהו, תחזיר התיבה ערך LB_ERR(-1).

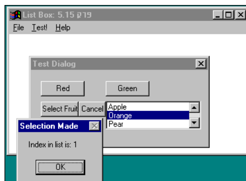
כדי להדגים את העיבוד של הודעת בחירה מתיבת רשימה, הוסף את משפטי Case הבאים למשפט ה-Switch הפנימי ((Switch(LOWORD(wParam בפונקציה ()TestDlgProc. כל פעם שנעשית בחירה באמצעות לחיצה כפולה על פריט כלשהו, תופיע תיבת הודעה ובה יוצג מספר האינדקס של הפריט שנבחר. אם המשתמש ילחץ על לחצן "Select Fruit", תדווח תיבת ההודעה גם על הפריט שנבחר.

```
case ID_LB1: /* process a list box LBN_DBLCLK */
   // see if user made a selection
   if(HIWORD(wParam)==LBN_DBLCLK)
   {
      i = SendDlgItemMessage(hDlg, ID_LB1,
                    LB_GETCURSEL, 0, 0L);  // get index
      sprintf(str, "Index in list is: %d", i);
      MessageBox(hDlg, str, "Selection Made", MB_OK);
   }
   break;
```

```
case IDD_SELFRUIT: /* Select Fruit has been pressed */
   i = SendDlgItemMessage(hDlg, ID_LB1,
                          LB_GETCURSEL, 0, 0L);  // get index
   if(i > -1)
      sprintf(str, "Index in list is: %d", i);
   else
      sprintf(str, "No Fruit Selected");
   MessageBox(hDlg, str, "Selection Made", MB_OK);
   break;
```

תיבת רשימה - התוכנית השלמה נמצאת בתיקיה Chap05**ListBox**

פלט לדוגמה מתוכנית זו מופיע בתרשים 5.3.



תרשים 5.3: פלט לדוגמה הכולל תיבת רשימה

5.16 כיצד להוסיף תיבת עריכה

הפקד האחרון שנציג בפרק זה הוא **תיבת העריכה** (Edit Box). תיבות עריכה שימושיות במיוחד, כי הן מאפשרות למשתמש להקליד מחרוזות תווים לבחירתו. לפני שתוכל להשתמש בתיבת עריכה, עליך להגדיר אותה בקובץ המשאבים שלך. עבור דוגמה זו, שנה את הקובץ **EditBox.rc** כך (ראה בתיקיה Chap05\EditBox):

```
TESTDIALOG DIALOG DISCARDABLE  20, 20, 178, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
```

```
BEGIN
    DEFPUSHBUTTON   "Red",IDD_RED,10,10,44,14
    PUSHBUTTON      "Green",IDD_GREEN,75,10,44,14
    PUSHBUTTON      "Cancel",IDCANCEL,49,29,27,14
    EDITTEXT        IDC_EDIT1,80,28,96,39,ES_AUTOHSCROLL
END
```

גירסה זו מוסיפה לתיבה לחצן בשם Edit OK, שתפקידו להודיע לתוכנית שסיימת את עריכת הטקסט בתיבת העריכה. היא גם מוסיפה לתוכנית את תיבת הרשימה עצמה. ID של תיבת הרשימה הוא ID_EB1. הגדרה זו גורמת ליצירת תיבת עריכה תקנית.

לפניך ההגדרה הכללית של משפט EDITTEXT :

```
EDITTEXT EDID, X, Y, Width, Height [,Style]
```

בהגדרה זו, EDID הוא הערך המזהה את תיבת העריכה. הפינה השמאלית-העליונה של התיבה תמוקם בנקודה X,Y, וגודלה יהיה מכפלת הפרמטר Width בפרמטר Height. הפרמטר Style קובע את הסגנון של תיבת הרשימה (הערכים לפרמטר זה הם אותם ערכים המופיעים בטבלה 5.1).

כעת, הוסיף את הגדרת המאקרו הבאה לקובץ EditBox.h :

```
#define ID_EB1        107
```

תיבות עריכה מכירות הודעות רבות, ואף מסוגלות להפיק כמה סוגי הודעות. למטרות דוגמה זו אין צורך שהתוכנית תגיב להודעות. כפי שיתברר לך, תיבות עריכה מבצעות את פונקציית העריכה בעצמן. כדי לערוך טקסט אין צורך בקשר כלשהו עם התוכנית. התוכנית תחליט מתי מתאים לה לקלוט את תכולת תיבת העריכה.

כדי לקלוט את התכולה הנוכחית של תיבת עריכה, השתמש בפונקציית API בשם GetDlgItemText(), שהגדרתה הכללית היא :

```
UINT GetDlgItemText(HWND hDlg, int nID, LPSTR lpstr, int nMax);
```

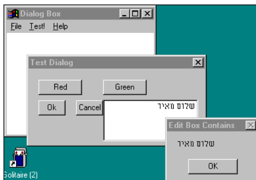
הפונקציה גורמת לתיבת העריכה להעתיק את התכולה הנוכחית של התיבה אל המחרוזת שהמצביע lpstr מצביע עליה. hDlg מכיל את הידית של תיבת הדו-שיח. הפרמטר nID מכיל את המספר המזהה של תיבת העריכה. המספר המירבי של תווים שיש להעתיק מוגדר על ידי הערך nMax. הפונקציה מחזירה את אורך המחרוזת.

כדי להוסיף תיבת עריכה לתוכנית שלנו, הוסף את משפט Case הבא למשפט Switch של הפונקציה TestDlgProc(). כל פעם שהמשתמש ילחץ על הלחצן Edit OK, יופיע על המסך חלון הודעה, שיכיל את הטקסט הקיים כרגע בתיבת העריכה.

```
case IDOK: /* edit box OK button selected
             display contents of the edit box */
      GetDlgItemText(hDlg, ID_EB1, str, 80);
      MessageBox(hDlg, str, "Edit Box Contains", MB_OK);
      return 1;
```

המאקרו IDOK הוא ערך מובנה, המוגדר על ידי הכללת הקובץ Windows.H בתוכנית.

בתרשים 5.4 מופיע פלט לדוגמה הנוצר על ידי תיבת העריכה.



תרשים 5.4: פלט לדוגמה הכולל טקסט מתוך תיבת העריכה.

פרק 6

ממשק התקן גרפי

6.1 ממשק ההתקן הגרפי (Graphics Device Interface)

ממשק ההתקן הגרפי - GDI (Graphics Device Interface) הוא שם כולל לקבוצת פונקציות תיקיה, אשר מספקות ליישומי Windows ממשק גרפי עבור מסך ומדפסות, שאינו תלוי בהתקנים. ממשק ההתקן הגרפי הוא שכבת ביניים בין היישום לבין סוגי החומרה השונים. ממשק ההתקן הגרפי משחרר את התוכניתן מעיסוק בכל סוג של התקן בצורה ישירה, בכך ש"תפקיד" זה מוטל על ממשק ההתקן הגרפי, אשר "פותר" עבוד התכניתן את בעיית ההבדלים בחומרה. יישום Windows מתוכנן היטב מתבצע באותה צורה על כל סוגי החומרה הנוכחיים ויפעל כראוי גם בכל חומרה חדשה שתיווצר בעתיד, מכיון שממשק ההתקן הגרפי "יפתור" כל שוני. למותר לציין, שהממשק צריך להתעדכן מעת לעת, כדי לאפשר את הגישור בין יישום המשתמש לבין סוגי החומרה החדשים.

כל פונקציות ממשק ההתקן הגרפי ב- Win32 משתמשות בערכים בני 32 סיביות עבור קואורדינטות ממשק ההתקן הגרפי; אך ב- Windows 9x ו- Win32, מערכת ההפעלה מתעלמת ממילת הסדר-הגבוה, דבר שבפועל גורם לקבלת ערך בן 16 סיביות עבור הקואורדינטות. רק ב- Windows NT יכולים היישומים להשתמש בערך המלא בן 32 סיביות.

6.2 כדאיות השימוש בממשק התקן גרפי

כפי שאולי דמיינת לעצמך, התוכניות צריכות להשתמש בממשק התקן גרפי כדי לייצר פלט במובן הכוללני, ולא להשתמש בפלט מבוסס טקסט או בפלט שאינו גרפי. יש סיבות רבות לשימוש בממשק התקן גרפי כדי לנהל את תכולת החלון. להלן כמה מהן.

- ☆ אפשר להשתמש באותו **הקשר התקן** (device context) עבור ההתקנים רבים.
- ☆ אפשר לפרמט פלט בהקשר ההתקן, לפני שליחתו אל ההתקן.
- ☆ אפשר לנהל גרפיקה ומידע חזותי אחר בחלון על ידי שימוש בהקשר התקן.
- ☆ אפשר לשלוט בצורה שבה נראה (או מופיע) החלון (אחרי גלילה, שינוי גודל, וכדומה) בקלות רבה יותר, על ידי שימוש בהקשר התקן.
- ☆ באפשרות התוכניות לשלוח בקלות אל מדפסת או אל הקשר התקן אחר כל פלט שמוקם קודם לכן בהקשר התקן של חלון.

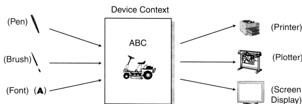
רבות מהתוכניות אשר הוצגו לפניך בספר זה השתמשו בהקשרי התקן, כדי לנהל נתונים בחלון התוכנית. בסעיפים הבאים, נתמקד מקרוב ביתרונות ובשימושים של הקשרי ההתקן.

6.3 להבין יותר טוב את הקשר ההתקן

הכלי הבסיסי ש-Windows משתמשת בו כדי לספק ליישום אי תלות בהתקן, הוא **הקשר ההתקן** (Device Context) ובקצרה - DC. הקשר ההתקן הוא מבנה פנימי, אשר Windows מנצלת להעברת נתונים להתקן פלט. במקום לשלוח פלט ישירות אל החומרה, היישום שולח את הפלט אל הקשר ההתקן. Windows, ולא התוכנית שלך, שולחת אחר כך את הפלט אל החומרה.

הקשר ההתקן תמיד מכיל **עט** (Pen) אחד כדי לצייר קווים, **מברשת** (Brush) אחת כדי למלא שטחים, **גופן** (Font) אחד לפלט של תווים, וסדרת ערכים אחרים לשליטה בהתנהגות הקשר ההתקן. אם היישום דורש גופן שונה, על היישום לבחור את הגופן לתוך הקשר ההתקן לפני הצגת הטקסט. בחירת גופן חדש אינה משנה את הטקסט הקיים בשטח הלקוח של החלון.

אפשר לדמות את הממשק שמספק הקשר ההתקן, לסכימה שמוצגת בתרשים 6.1 שלהלן.



תרשים 6.1: המודל הלוגי של הקשר ההתקן.

6.4 הקשרי התקן פרטיים

בדרך כלל, יישומים משתפים ומקבלים הקשרי התקן מיד לפני השימוש בהם, ומשחררים אותם מיד אחרי השימוש. ניהול הקשרי התקן לפרקי זמן קצרים הינו טוב ביישומים שאינם משתמשים לעיתים תכופות בהקשר ההתקן. יישום שזקוק לשירותי הקשר ההתקן באופן מתמיד, יכול ליצור חלון עם הקשר ההתקן הפרטי ששייך לו, על ידי הגדרת **סגנון המחלקה** (Class Style) CS_OWNDC במחלקת הגדרת החלון. לאחר הגדרת הסגנון CS_OWNDC, הקשר ההתקן מתקיים למשך החיים של החלון. היישום ממשיך להשתמש בפונקציה GetDC כדי לקבל ידית להקשר ההתקן, אבל אינו צריך להשתמש בפונקציה ReleaseDC לאחר שהוא מסיים את העיבוד בהקשר ההתקן (מכיון שהקשר ההתקן הינו "פרטי" שלו). כאשר אתה משתמש בהקשר ההתקן הפרטי בתוכניות שמשנות את הערכים המוגדרים בהקשר ההתקן, אתה גורם לשינויים, כמו צבעי טקסט חדשים, עטים, ומברשות. שינויים אלה נשארים בתוקף עד שהתוכנית משנה אותם פעם נוספת.

6.5 נקודות מוצא ומדידת מרחק

יש להקשר ההתקן שני **מצבי מיפוי** (Mapping Mode) מיוחדים, MM_ISOTROPIC ו-MM_ANISOTROPIC, שאינם מוגבלים לגודל קבוע. שני מצבי מיפוי אלה משתמשים בשני אזורים מלבניים, **החלון** (Window) **ואזור התצוגה** (Viewport), כדי לגזור גורם דירוג (Scaling Factor) **והכוונה** (Orientation). החלון מוצג ב**קואורדינטות לוגיות** (Logical Coordinates) ואזור התצוגה - ב**קואורדינטות פיזיות** (Physical Coordinates). יחד הם קובעים איך מערכת ההפעלה ממפה **יחידות לוגיות** (Logical Units) ל**יחידות פיזיות** (Physical Units). הן החלון והן אזור התצוגה מכילים **נקודת מוצא** (Origin), **ערך x** (X-Extent) **וערך y** (Y-Extent). **נקודת המוצא** היא נקודה אשר מתארת את אחת מפינות מלבן התצוגה. **נקודת המוצא של אזור התצוגה** (Viewport Origin) היא **היסט** (Offset, או מרחק יחסי) מ**נקודת המוצא של החלון** (Window Origin); **X-Extent** הוא המרחק האופקי מנקודת המוצא עד לזווית שמשמאלה; **Y-Extent** הוא המרחק האנכי מנקודת המוצא עד לזווית שמשמאלה.

Windows מגדירה גורם דירוג אופקי על ידי חלוקת X-Extent של אזור התצוגה ב- X-Extent של החלון. Windows מגדירה גורם דירוג אנכי על ידי חלוקת Y-Extent של אזור התצוגה Y-Extent של החלון. גורמי דירוג אלה מגדירים את מספר היחידות הלוגיות אשר Windows ממפה אותם למספר **פיקסלים** (Pixels). בנוסף לקביעת גורמי הדירוג, החלון ואזור התצוגה קובעים גם את הכיוון של אובייקט.

6.6 קבלת הקשר התקן אל החלון

התוכניות צריכות להשתמש בהקשרי התקן, כדי לבנות תצוגות גרפיקה וטקסט בשני ההתקנים, בהתקן חלון ובהתקן מדפסת. באפשרות התוכניות להשתמש בפונקציה GetDC או GetDCEx כדי להשיג הקשר התקן עבור חלון. הפונקציה GetDCEx משיגה את הידית של הקשר ההתקן לתצוגה (DC) עבור החלון המוגדר. באפשרות התוכניות להשתמש בהקשר ההתקן לתצוגה על ידי קריאות עוקבות לפונקציות ממשק התקן גרפי, כדי לצייר בשטח הלקוח. הפונקציה GetDCEx היא הרחבה לפונקציה GetDC אשר מאפשרת ליישום שליטה רבה יותר על הדרך והזמן לביצוע גזירה (Clipping) בשטח הלקוח. את הפונקציה GetDCEx כותבים בתוכנית כמו בהגדרה שלהלן:

```
HDC GetDCEx(
  HWND hWnd,            // handle of window
  HRGN hrgnClip,        // handle of clip region
  DWORD flags,          // device-context creation flags
);
```

הפרמטר hWnd מזהה את החלון שבו יבוצע הציור. הפרמטר hrgnClip מגדיר **אזור גזירה** (Clipping Region) אשר אולי תחבר אותו עם **האזור הנראה** (Visible Region) של חלון הלקוח. הפרמטר Flags מגדיר כיצד היישום יוצר את הקשר ההתקן. הפרמטר Flags יכול להכיל צירוף כלשהו של הערכים שמפורטים בטבלה 6.1.

במקרה שהקשר ההתקן לתצוגה אינו שייך למחלקת חלון, היישום חייב לקרוא לפונקציה ReleaseDC כדי לשחרר את הקשר ההתקן אחרי צביעה. מכיון שרק חמישה הקשרי התקן משותפים זמינים לתוכניות בכל זמן נתון, כישלון בשחרור הקשר ההתקן יכול למנוע מתוכניות אחרות את הגישה להקשר ההתקן. שתי הפונקציות GetDC ו-GetDCEx מחזירות את הקשר ההתקן ששייך למחלקת החלון, אם התוכנית מגדירה CS_CLASSDC ,CS_OWNDC, או CS_PARENTDC כסגנון במבנה WNDCLASS כאשר התוכנית רשמה את המחלקה.

טבלה 6.1: הערכים האפשריים של הפרמטר **flags**.

ערך	פירוש
DCX_WINDOW	מחזיר את הקשר ההתקן בהתאם למלבן החלון, ולא בהתאם למלבן הלקוח.
DCX_CACHE	מחזיר את הקשר ההתקן מהמטמון (Cache), ולא מחלון OWNDC או CLASSDC. באופן עקרוני, ערך זה עוקף ודורס (Overrides) את CS_OWNDC ואת CS_CLASSDC.
DCX_PARENTCLIP	משתמש באזור הנראה (Visible Region) של חלון האב. ערך זה מתעלם מסגנונות האב WS_CLIPCHILDREN ו-CS_PARENTDC. GetDCEx קובעת את נקודת המוצא של אזור הקשר ההתקן לפינה העליונה משמאל של החלון המוגדר על ידי hWnd.

ערך	פירוש
DCX_CLIPSIBLINGS	אינו כולל את האזורים הנראים של כל החלונות השכנים שמעל לחלון המוגדר על ידי hWnd.
DCX_CLIPCHILDREN	אינו כולל את האזורים הנראים של כל חלונות הבן שמתחת לחלון המוגדר על ידי hWnd.
DCX_NORESETATTRS	אינו מאפס את המאפיינים של הקשר התקן זה למאפייני ברירת המחדל, כאשר התוכנית משחררת הקשר התקן זה.
DCX_LOCKWINDOWUPDATE	מאפשר לתוכנית ולמשתמש לצייר בהקשר ההתקן, אפילו אם יש קריאה ל-LockWindowUpdate ; אחרת, הדבר יכול לגרום להוצאת חלון זה. משתמשים בערך זה כדי לאפשר לתוכניות לצייר בזמן פעולת עקיבה (Tracking Operation).
DCX_EXCLUDERGN	אינו כולל את אזור הגזירה שמוגדר על ידי hrgnClip מהאזור הנראה של הקשר ההתקן שמוחזר על ידי GetDCEx.
DCX_INTERSECTRGN	חיתוך (Intersect) אזור הגזירה שמוגדר על ידי hrgnClip עם האזור הנראה של הקשר ההתקן שמוחזר על ידי GetDCEx.
DCX_VALIDATE	כאשר ערך זה מוגדר עם DCX_INTERSECTUPDATE, הדבר גורם להקשר ההתקן שמוחזר על ידי GetDCEx להיות זמין בשלמותו. השימוש בפונקציה זאת יחד עם DCX_INTERSECTUPDATE ו-DCX_VALIDATE זהה לשימוש בפונקציה BeginPaint.

כדי להבין יותר טוב את פעולת הפונקציה GetDCEx, התבונן בתוכנית **Draw_Hollow**, שבתקליטור המצורף לספר זה (בתיקיה Chap06). התוכנית משתמשת בפונקציה GetDCEx כדי להשיג את הקשר ההתקן לשטח הלקוח של החלון, חוץ מאזור מסוים. התוכנית מציירת אחר כך מלבן אפור בשטח הלקוח, ללא האזור הפנימי (הלבן).

6.7 יצירת הקשר התקן למדפסת

התוכניות צריכות להשתמש בהקשרי התקן כדי לצייר פלט בהתקן. בעוד שיש ל-Windows הקשרי התקן מובנים (פרטי או ציבורי, Private או Public), להתקני פלט אחרים אין הקשרים קיימים. על התוכניות להשתמש בפונקציה CreateDC כדי ליצור הקשר התקן אל ההתקן ושימוש בשם שמוגדר על ידי המתכנת. את הפונקציה CreateDC כותבים בתוכנית כפי שמוצג להלן.

```
HDC CreateDC(
  LPCTSTR lpszDriver,   // pointer to string
                        // specifying driver name
  LPCTSTR lpszDevice,   // pointer to string
                        // specifying device name
  LPCTSTR lpszOutput,   // do not use; set to NULL
  CONST DEVMODE *lpInitData       // pointer to optional
                        // printer data
);
```

הפונקציה CreateDC מקבלת את הפרמטרים שמפורטים בטבלה 6.2.

טבלה 6.2: הפרמטרים של הפונקציה **CreateDC**.

פרמטר	תיאור
lpszDriver	יישומים שנכתבו עבור גרסאות קודמות של Windows משתמשים בפרמטר זה, כדי להגדיר את שם הקובץ (בלי סיומת) של מנהל ההתקן. ב- Windows 9x ויישומים מבוססי Win32, CreateDC מתעלמת מפרמטר זה, שצריך להיות NULL, עם יוצא מן הכלל: אולי תרצה להשיג הקשר התקן לתצוגה על ידי הגדרת המחרוזת DISPLAY המסתיימת ב-NULL. כאשר פרמטר זה שווה DISPLAY, כל שאר הפרמטרים חייבים להיות NULL. בסביבת Windows NT, הפרמטר lpszDriver מצביע למחרוזת תווים המסתיימת ב-NULL אשר מגדירה DISPLAY עבור מנהל תצוגה, או שם מנהל התקן מדפסת, שבדרך כלל הוא WINSPOOL.
lpszDevice	מצביע למחרוזת תווים המסתיימת ב-NULL שמגדירה את שם התקן הפלט המסוים שהתוכנית משתמשת בו. מנהל הדפסה מראה את שם התקן הפלט, כמו לדוגמה "Epson FX-80", שהינו שם פנימי של Windows, ולא בהכרח שם סוג המדפסת. אתה חייב להשתמש בפרמטר lpszDevice.
lpszOutput	Windows מתעלמת מפרמטר זה. אל תשתמש בו ביישומי Win32. יישומים מבוססי Win32 צריכים להציב בפרמטר זה את הערך NULL, מכיון ש-lpszOutput קיים כדי לספק תאימות עם יישומים שנכתבו עבור גרסאות מוקדמות של Windows.
lpInitData	מצביע למבנה מסוג DEVMODE שמכיל עבור מנהל ההתקן נתוני אתחול ייחודיים להתקן. הפונקציה DocumentProperties מקבלת מבנה זה וממלאת אותו בהתאם להתקן המוגדר. הפרמטר lpInitData חייב להיות NULL אם מנהל ההתקן משתמש באתחול ברירת המחדל (אם ישנה), שמוגדר על ידי המשתמש.

אם הפונקציה CreateDC מצליחה, הערך המוחזר הוא ידית להקשר התקן עבור ההתקן המוגדר; אם CreateDC נכשלת, הערך המוחזר הוא NULL.

כמו שראית בטבלה 6.2, הפונקציה CreateDC מצפה בפרמטר האחרון שלה למצביע למבנה מסוג DEVMODE אשר מכיל עבור מנהל ההתקן את נתוני האתחול הייחודיים להתקן.

Win32 API מגדירה את המבנה DEVMODE כפי שמוצג להלן:

```
typedef struct _devicemode {
  TCHAR              dmDeviceName[32];
  WORD               dmSpecVersion;
  WORD               dmDriverVersion;
  WORD               dmSize;
  WORD               dmDriverExtra;
  DWORD              dmFields;
  short              dmOrientaion;
  short              dmPaperSize;
  short              dmPaperLength;
  short              dmPaperWidth;
  short              dmScale;
  short              dmCopies;
  short              dmDefaultSource;
  short              dmPrintQuality;
  short              dmColor;
  short              dmDuplex;
  short              dmYResolution;
  short              dmTTOption;
  short              dmCollate;
  TCHAR              dmFormName[32];
  WORD               dmUnusedPadding;
  USHORT             dmBitsPerPel;
  DWORD              dmPelsWidth;
  DWORD              dmPelsHeight;
  DWORD              dmDisplayFlags;
  DWORD              dmDisplayFrequency;
} DEVMODE;
```

מבנה הנתונים DEVMODE מכיל מידע על אתחול ההתקן וסביבת העבודה של המדפסת.

טבלה 6.3 מפרטת את האיברים של מבנה הנתונים DEVMODE.

טבלה 6.3: איברי המבנה DEVMODE.

איבר	תיאור
dmDeviceName	מגדיר את שם ההתקן שמנהל ההתקן תומך בו (לדוגמה, "PCL/HP LaserJet" במקרה של PCL/HP LaserJet®). מחרוזת זאת היא מיוחדת בין מנהלי ההתקנים.
dmSpecVersion	מגדיר את מספר הגירסה של תיאור נתוני האתחול שמבוסס עליהם מבנה הקשר ההתקן.
dmDriverVersion	מגדיר את מספר גרסת מנהל ההתקן של המדפסת, אשר מוגדר על ידי מפתח מנהל ההתקן.
dmSize	מגדיר את הגודל, בבתים, של המבנה DEVDEMO. אם יישום מנהל רק את חלק הנתונים העצמאי של מנהל ההתקן, באפשרות היישום להשתמש באיבר זה כדי לקבוע את אורך המבנה מבלי שיצטרך להתחשב בגרסאות שונות.
dmDriverExtra	מכיל את מספר הבתים לנתונים פרטיים של מנהל ההתקן אשר עוקבים למבנה DEVDEMO. אם מנהל ההתקן אינו משתמש במידע ייחודי להתקן, קבע ערך זה לאפס.
dmFields	מגדיר איזה מהאיברים שנשארו במבנה DEVDEMO אותחל כבר. סיבית אפס (מוגדרת כ-DM_ORIENTATION) מתאימה ל-dmOrientaion ; סיבית 1 (מוגדרת כ-DM_PAPERSIZE), אשר מגדירה את dmPaperSize, וכך הלאה. מנהל התקן המדפסת תומך רק באיברים שמתאימים לטכנולוגיית המדפסת.
dmOrientaion	בוחר את כיוון הדף. איבר זה יכול להיות ערך אחד משני הערכים הבאים : (1) DMORIENT_PORTRAIT (לאורך), או (2) DEMORIENT_LANDSCAPE (לרוחב).
dmPaperSize	בוחר את גודל הדף להדפסה. אפשר לקבוע איבר זה לאפס, אם רוחב הדף ואורכו נקבעים על ידי האיברים dmPaperLength ו-dmPaperWidth. אחרת, באפשרותך לקבוע את האיבר dmPaperSize לאחד מרשימת הערכים המובנים שמפורטים בטבלה 6.4.
dmPaperLength	עוקף את אורך הדף שמוגדר על ידי האיבר dmPaperSize, עבור גודל דף שמותאם אישית או עבור התקנים, כמו מדפסות סיכות, שיכולות להדפיס על דף שאורכו אינו מוגבל. ערך זה, יחד עם שאר הערכים שבמבנה זה אשר מגדירים אורך פיזי, הם ביחידות של עשירית המילימטר.

איבר	תיאור
dmPaperWidth	עוקף את רוחב הדף שמוגדר על ידי האיבר dmPaperSize.
dmScale	מגדיר את הגורם שבו המבנה DEVDEMO מדרג (Scales) את הפלט המודפס. איבר זה מדרג את גודל הדף המודפס ביחס לגודל הפיזי של הדף, ביחס של dmScale/100. לדוגמה, גודל דף קוּוארטו (Letter-Sized) עם ערך dmScale ששווה ל- 50 מכיל נתונים כמו דף בגודל 17x22 אינץ' (כ- 43x56 ס"מ), מפני שפלט הטקסט והגרפיקה יהיו במחצית האורך והרוחב המקוריים שלהם.
dmCopies	בוחר את מספר ההעתקים שיודפסו אם ההתקן תומך בריבוי העתקים.
dmDefaultSource	שמור - חייב להיות אפס.
dmPrintQuality	מגדיר את הרזולוציה של המדפסת. יש ארבעה ערכים מובנים שאינם תלויים בהתקן: DMRES_HIGH DMRES_MEDIUM DMRES_LOW DMRES_DRAFT אם איבר זה מכיל ערך חיובי שאינו קבוע (Non-Constant), אז הערך מגדיר את מספר הנקודות לאינץ' (DPI – Dots Per Inch) ולכן הופך לתלוי בהתקן.
dmColor	בחירה בין הדפסה בצבע לבין הדפסה בצבע אחד במדפסות צבע. הערכים שלהלן הם ערכים אפשריים לפרמטר זה: DMCOLOR_COLOR DMCOLOR_MONOCHROME
dmDuplex	בוחר הדפסה דו-צידית אוטומטית (Duplex), או הדפסה דו-צידית רגילה (Double Sided, עם הפיכה ידנית של הנייר), עבור מדפסות שיכולות להדפיס בשיטת Duplex. הערכים שלהלן אפשריים לפרמטר זה: DMDUP_SIMPLEX DMDUP_HORIZONTAL DMDUP_VERTICAL
dmYResolution	מגדיר רזולוציית ציר Y (Y-Resolution) של המדפסת בנקודות לאינץ'. אם המדפסת מאתחלת איבר זה, האיבר dmPrintQuality מגדיר את רזולוציית ציר X (X-Resolution) של המדפסת, בנקודות לאינץ'.

איבר	תיאור
dmTTOption	מגדיר איך המדפסת צריכה להדפיס גופני ®TrueType. איבר זה יכול להכיל אחד מהערכים שלהלן: DMTT_BITMAP מדפיס גופני TrueType כגרפיקה. זוהי פעולת ברירת המחדל במדפסות סיכות. DMTT_DOWNLOAD גופני TrueType נטענים כגופנים רכים (Soft Fonts). זוהי פעולת ברירת המחדל במדפסות היולט-פקרד (Hewlett Packard) אשר משתמשות בשפת הבקרה למדפסת – PCL (Printer Control Language). DMTT_SUBDEV מחליף הגופנים של ההתקן לגופני TrueType. זוהי פעולת ברירת המחדל במדפסות ®PostScript.
dmCollate	מגדיר אם המדפסת צריכה להשתמש באיסוף עותקים, כאשר מדפיסים עותקים רבים. השימוש ב-DMCOLLATE_FALSE מאפשר מהירות גבוהה ופלט יעיל יותר, מכיון שהנתונים נשלחים למדפסת רק פעם אחת עבור כל דף מודפס, ללא תלות במספר העותקים הנדרש. התוכנית שולחת את הדף למדפסת ומציינת את מספר הפעמים שדרוש להדפיסו. האיבר dmCollate יכול להיות אחד מהערכים שלהלן: DMCOLLATE_TRUE - איסוף כאשר מדפיסים העתקים רבים. DMCOLLATE_FALSE - ללא איסוף, כאשר מודפסים העתקים רבים.
dmFormName	מגדיר את השם של התבנית (Form) שצריך להשתמש בה (לדוגמה, מכתב רגיל או מכתב רשמי). אפשר לקבל קבוצה שלמה של שמות באמצעות הפונקציה EnumForms של Windows.
dmUnusedPadding	מיישר את המבנה לגבול מילה כפולה (DWORD). אינך צריך להשתמש באיבר זה או להתייחס אליו. מיקרוסופט שומרת את השם והשימוש שלו, והדבר עלול להשתנות בגירסאות עתידיות של Windows.
dmBitsPerPel	מגדיר בסיביות לפיקסל את רזולוציית הצבע של התקן התצוגה. לדוגמה, 4 סיביות עבור 16 צבעים, 8 סיביות עבור 256 צבעים, או 16 סיביות עבור 65536 צבעים.
DmPelsWidth	מגדיר את הרוחב, בפיקסלים, של משטח ההתקן הנראה.

איבר	תיאור
dmPelsHeight	מגדיר את הגובה, בפיקסלים, של משטח ההתקן הנראה.
dmDisplayFlags	מגדיר את מצב התצוגה של ההתקן. הערכים שלהלן הם דגלים תקפים עבור האיבר dmDisplayFlags: DM_GRAYSCALE מגדיר שהתצוגה אינה התקן צבעוני. אם לא קובעים דגל זה, התוכנית מניחה צבע. DM_INTERLACED מגדיר מצב תצוגה בדילוג (Interlaced). אם לא קובעים דגל זה, התוכנית מניחה שמצב התצוגה הוא תצוגה בסריקה רציפה (Non_Interlaced).
dmDisplayFrequency	מגדיר את התדירות, בהרץ (מחזורים לשנייה), של התקן התצוגה במצב עבודה מסוים.

כמו שמפורט בטבלה 6.3, הפרמטר dmPaperSize מקבל קבועים מובנים שמתאימים לגודלי נייר נפוצים ובינלאומיים. טבלה 6.4 מפרטת חלק מגודלי הנייר המקובלים.

טבלה 6.4: ערכים מקובלים לגודל הנייר.

ערך	גודל הנייר
DMPAPER_LETTER	מכתב (Letter), 1/2 8 X 11 אינץ'
DMPAPER_LEGAL	ליגאל (Legal), 1/2 8 X 14 אינץ'
DMPAPER_A4	גיליון A4, 210 X 297 מילימטר
DMPAPER_LEDGER	Ledger, 17 X 11 אינץ'
DMPAPER_STATEMENT	Statement, 1/2 5 X 1/2 8 אינץ'
DMPAPER_EXECUTIVE	Executive, 1/4 7 X 1/2 10 אינץ'
DMPAPER_FOLIO	פוליו (Folio), 1/2 8 X 13 אינץ'
DMPAPER_QUARTO	קווארטו (Quarto), 215 X 275 מילימטר
DMPAPER_11X17	גיליון 11 X 17 אינץ'
DMPAPER_ENV_10	מעטפות #10, 1/8 4 X 1/2 9 אינץ'
DMPAPER_FANFOLD_US	US Std Fanfold, 7/8 14 X 11 אינץ'
DMPAPER_FANFOLD_LGL_GERMAN	ליגאל גרמני Fanfold, 1/2 8 X 13 אינץ'

הנתונים הפרטיים של מנהל ההתקן עוקבים אחרי האיבר dmDisplayMode. האיבר dmDriverExtra מגדיר את מספר הבתים של נתונים פרטיים. היישומים שנכתבו עבור גרסאות קודמות של Windows משתמשים בפרמטר lpszOutput כדי להגדיר שם יציאה, או להדפיס לקובץ. היישומים מבוססי Win32 אינם צריכים להגדיר שם יציאה. היישומים מבוססי Win32 יכולים להדפיס לקובץ על ידי קריאה לפונקציה StartDoc עם המבנה DOCINFO, אשר הפרמטר lpszOutput שלו מגדיר את המסלול של שם קובץ הפלט. כאשר אינך צריך יותר את הקשר ההתקן, קרא לפונקציה DeleteDC למחוק אותו.

כדי להבין יותר טוב את פעולת הפונקציה CreateDC, התבונן בתוכנית **Print_File**, שבתקליטור המצורף לספר זה (בתיקיה Books\59285\chap06). התוכנית מדפיסה שורה אחת של טקסט במדפסת, כאשר המשתמש בוחר באפשרות Test! מהתפריט. CreateDC יוצרת את הקשר ההתקן למדפסת. כמו שכתוב בקוד שבתקליטור, שם המדפסת הוא "Kyosera FS-680", אבל באפשרותך לשנות אותו, כדי שיתאים למנהל ההתקן שנמצא במערכת שלך. תוכל להשתמש בפונקציה EnumPrinters כדי לקבוע את שם המדפסת שהתקנת במחשב שלך. הפונקציה WndProc מכילה את הקוד המעשי שמטפל במשימות של התוכנית **Print_File**.

6.8 CreateCompatibleDC - יצירת הקשר התקן בזיכרון

בסעיפים קודמים למדת שאפשר להשתמש בהקשרי התקן בתוכניות כדי לייצר פלט לתצוגה ולמדפסות. עם זאת, התוכניות אינן יכולות לצייר ישר לתוך הקשר ההתקן. הפונקציה CreateCompatibleDC יוצרת הקשר התקן בזיכרון אשר מתאים להתקן שצוין. לפני שיישום יכול להשתמש בהקשר התקן בזיכרון לפעולות ציור, אתה חייב לבחור **מפת סיביות** (Bitmap) ברוחב ובגובה הנכונים בתוך הקשר ההתקן. אחרי בחירת מפת סיביות, באפשרותך להשתמש בהקשר ההתקן כדי להכין דמויות שהתוכנית תעתיק אל המסך או המדפסת. בכל פעם שהתוכניות עובדות עם מפות סיביות (אשר תלמד עליהן בסעיפים הבאים), התוכנית ממקמת תחילה את מפת הסיביות בהקשר התקן בזיכרון ואחר כך מעתיקה אותה אל הקשר התקן מסוים. את הפונקציה CreateCompatibleDC כותבים בתוכנית כפי שמוצג להלן:

```
HDC CreateCompatibleDC(HDC hdc);
```

הפרמטר hdc מזהה את הקשר ההתקן. אם הידית hdc שווה NULL, הפונקציה CreateCompatibleDC יוצרת הקשר התקן בזיכרון שמתאים למסך הנוכחי של היישום. אם הפונקציה CreateCompatibleDC מצליחה, הערך המוחזר הוא ידית להקשר התקן בזיכרון; ואם CreateCompatibleDC נכשלת, הערך המוחזר הוא NULL.

אפשר להשתמש בפונקציה CreateCompatibleDC רק עם התקנים שתומכים **בפעולות רשת** (Raster Operations). היישום יכול לקבוע אם ההתקן תומך בפעולות רשת על ידי קריאה לפונקציה GetDeviceCaps. כאשר אין צורך יותר בהקשר ההתקן שבזיכרון, צריך לקרוא לפונקציה DeleteDC כדי למחוק אותו.

כדי להבין יותר טוב את פעולת הפונקציה CreateCompatibleDC, התבונן בתוכנית **Draw_Bitmap**, שבתקליטור המצורף לספר זה (בתיקיה chap06). התוכנית טוענת מפת סיביות וממקמת אותה בהקשר ההתקן שבזיכרון, ואחר כך מעתיקה אותה לשטח הלקוח של החלון. כרגיל, הפונקציה WndProc מכילה את הקוד המעשי של התוכנית **Draw_Bitmap**.

6.9 אחזור יכולות ההתקן

התוכניות משתמשות בהקשר התקן כדי לנהל טקסט וגרפיקה ביישומים שלך לשתי מטרות, לתצוגה על המסך ועבור פלט להתקנים, כמו מדפסות או תוויינים (Plotters). כאשר התוכנית חייבת להפיק פלט למדפסת או תוויין, יהיה דרוש לנו מידע על יכולות ההתקן לפני שנשלח אליו את הפלט. משתמשים בפונקציה GetDeviceCaps כדי לקבל את המידע הייחודי אודות ההתקן המוגדר. את הפונקציה GetDeviceCaps כותבים כמו בהגדרה שלהלן:

```
int GetDeviceCaps(HDC hdc, int nIndex);
```

הפרמטר hdc מזהה את הקשר ההתקן. הפרמטר nIndex מזהה את הפריט שצריך להחזיר. הפרמטר nIndex יכול להיות אחד הערכים שמפורטים בטבלה 6.5.

טבלה 6.5: ערכים אפשריים של הפרמטר **nIndex**.

אינדקס (או מפתח)	פירוש
DRIVERVERSION	גרסת מנהל ההתקן.
TECHNOLOGY	הטכנולוגיה של ההתקן; יכולה להיות כל ערך מהערכים שמפורטים בטבלה 6.6.
HORZSIZE	רוחב, במילימטרים, של המסך הפיזי.
VERTSIZE	גובה, במילימטרים, של המסך הפיזי.
HORZRES	רוחב, בפיקסלים, של המסך.
VERTRES	גובה, בקווי רשת, של המסך.
LOGPIXELSX	מספר הפיקסלים בכל אינץ' לוגי לרוחב המסך.
LOGPIXELSY	מספר הפיקסלים בכל אינץ' לוגי לגובה המסך.
BITSPIXEL	מספר סיביות הצבעים הסמוכים עבור כל פיקסל.
PLANES	מספר מישורי הצבע.
NUMBRUSHES	מספר המברשות בהתקן המסוים.
NUMPENS	מספר העטים בהתקן המסוים.

אינדקס (או מפתח)	פירוש
NUMFONTS	מספר הגופנים בהתקן המסוים.
NUMCOLORS	מספר הכניסות בטבלת הצבעים של ההתקן, אם יש להתקן עומק צבע (color depth) של לא יותר מ- 8 סיביות לכל פיקסל. עבור התקנים עם עומק צבע יותר גדול, GetDeviceCaps מחזירה 1-.
ASPECTX	רוחב יחסי של פיקסל ההתקן, שההתקן משתמש בו לציור קווים.
ASPECTY	גובה יחסי של פיקסל ההתקן, שההתקן משתמש בו לציור קווים.
ASPECTXY	רוחב האלכסון של פיקסל ההתקן, שההתקן משתמש בו לציור קווים.
PDEVICESIZE	שמור.
CLIPCAPS	דגל אשר מציין את יכולות הגזירה של ההתקן. אם ההתקן יכול לגזור למלבן, ערך הפרמטר הוא 1; אחרת, ערכו לאפס.
SIZEPALETTE	מספר הכניסות בלוח הצבעים (Palette) של המערכת. אינדקס זה תקף רק אם מנהל ההתקן מדליק את הסיבית RC_PALETTE באינדקס RASTERCAPS, והוא זמין רק אם מנהל ההתקן תואם את גרסת Windows 9x ומעלה.
NUMRESERVED	מספר הכניסות השמורות בלוח הצבעים של המערכת. אינדקס זה תקף רק אם מנהל ההתקן מדליק את הסיבית RC_PALETTE באינדקס RASTERCAPS, והוא זמין רק אם מנהל ההתקן תואם Windows 9x ומעלה.
COLORRES	רזולוציית הצבע הממשית של ההתקן, בסיביות לפיקסל. אינדקס זה תקף רק אם מנהל ההתקן מדליק את הסיבית RC_PALETTE באינדקס RASTERCAPS, והוא זמין רק אם מנהל ההתקן תואם Windows גירסה 9x ומעלה.
PHYSICALWIDTH	עבור התקני ההדפסה, זהו רוחב הדף הפיזי, ביחידות התקן. לדוגמה, למדפסת שהגדרותיה נקבעו להדפסה ב- 600dpi בדף שגודלו 11 x 8.5 אינץ', יש רוחב פיזי של 5100 יחידות התקן. שים לב לכך שהדף הפיזי כמעט תמיד גדול משטח ההדפסה שעל פניו, ולעולם אינו קטן ממנו.

אינדקס (או מפתח)	פירוש
PHYSICALHEIGHT	עבור התקני ההדפסה, זהו גובה הדף הפיזי, ביחידות ההתקן. לדוגמה, מדפסת שהגדרותיה נקבעו להדפסה ב- 600dpi בדף שגודלו 11 x 8.5 אינץ', יש גובה פיזי של 6600 יחידות התקן. שים לב לכך שהדף הפיזי כמעט תמיד גדול משטח ההדפסה שעל פניו, ולעולם אינו קטן ממנו.
PHYSICALOFFSETX	עבור התקני ההדפסה, המרחק ביחידות התקן מהשפה השמאלית של הדף הפיזי עד לשפה השמאלית של שטח ההדפסה. לדוגמה, מדפסת שהגדרותיה נקבעו להדפסה ב- 600dpi בדף שגודלו 11 x 8.5 אינץ', ואשר אינה יכולה להדפיס במרווח 0.25 האינץ' שבצדו השמאלי ביותר של הדף, יש היסט (Offset, או מרחק יחסי) פיזי אופקי של 150 יחידות התקן.
PHYSICALOFFSETY	עבור התקני ההדפסה, המרחק ביחידות התקן מהשפה העליונה של הדף הפיזי עד לשפה העליונה של שטח ההדפסה. לדוגמה, מדפסת שהגדרותיה נקבעו להדפסה ב- 600dpi בדף שגודלו 11 x 8.5 אינץ', ואשר אינה יכולה להדפיס במרווח 0.5 האינץ' העליון של הדף, יש היסט (Offset, או מרחק יחסי) פיזי אנכי של 300 יחידות התקן.
VREFRESH	עבור התקני תצוגה, תחת Windows NT, זהו קצב הרענון האנכי הנוכחי של ההתקן, במחזורים לשנייה (Hz). קצב רענון אנכי של 0 או 1 מייצג את ברירת המחדל של קצב רענון חומרת התצוגה (Display Hardware's Default Refresh Rate). ערך ברירת מחדל זה נקבע בדרך כלל על ידי מתגים שעל כרטיס התצוגה או על לוח האם של המחשב, או על ידי תוכנית תצורה, שאינה משתמשת בפונקציות התצוגה של Win32, כמו לדוגמה ChangeDisplaySettings.
DESKTOPHORZRES	רוחב, בפיקסלים, של שולחן העבודה הווירטואלי (Virtual Desktop), רק תחת Windows NT. ערך זה יכול להיות גדול מ-HORZRES כאשר ההתקן תומך בשולחן עבודה וירטואלי או בתצוגות מרובות.
DESKTOPVERTRES	גובה, בפיקסלים, של שולחן העבודה הווירטואלי (Virtual Desktop), רק תחת Windows NT. ערך זה יכול להיות גדול מ-VERTRES כאשר ההתקן תומך בשולחן עבודה וירטואלי, או בתצוגות מרובות.

אינדקס (או מפתח)	פירוש
BLTALIGNMENT	היישור המועדף של ההתקן עבור ציור אופקי, מיוצג על ידי פיקסלים רבים, רק תחת Windows NT. עבור ביצועי ציור טובים, התוכניות צריכות ליישר חלונות אופקית לפי כפולות של ערך זה. ערך אפס מציין שיש ל-Windows גישה מואצת להתקן, והקשרי התקן של ההתקן הזה יכולים להשתמש ביישור כלשהו.
RASTERCAPS	ערך אשר מציין את יכולות הרשת או הסריקה (Raster Capabilities) של ההתקן, כמו שמפורט בטבלה 6.7.
CURVECAPS	ערך אשר מציין את יכולות העקומה (Curve Capabilities) של ההתקן, כמו שמפורט בטבלה 6.8.
LINECAPS	ערך אשר מציין את יכולות הקו (Line Capabilities) של ההתקן, כמו שמפורט בטבלה 6.9.
POLYGONALCAPS	ערך אשר מציין את יכולות הפוליגון (Polygon Capabilities) של ההתקן, כמו שמפורט בטבלה 6.10.
TEXTCAPS	ערך אשר מציין את יכולות הטקסט (Text Capabilities) של ההתקן, כמו שמפורט בטבלה 6.11.

בטבלה 6.5 ניתן לראות שאם התוכנית דורשת מההתקן את טכנולוגיית מנהל ההתקן, הקריאה ל-GetDeviceCaps מחזירה אחד מערכי הדגלים שמפורטים בטבלה 6.6.

טבלה 6.6: ערכי הדגלים המוחזרים עבור סוגים אפשריים של טכנולוגיית מנהלי התקן.

ערך	פירוש
DT_PLOTTER	תוויין ווקטורי (Vector Plotter)
DT_RASDISPLAY	צג רשת / סריקה (Raster Display)
DT_RASPRINTER	מדפסת רשת / סריקה (Raster Printer)
DT_RASCAMERA	מצלמת רשת / סריקה (Raster Printer)
DT_CHARSTREAM	רצף תווים (Character Stream)
DT_METAFILE	קובץ-על (Metafile, או קובץ Metafile)
DT_DISPFILE	קובץ תצוגה (Display file)

חשוב לשים לב לכך שאם הפרמטר hdc מזהה את הקשר ההתקן של קובץ-על משופר (Enhanced Metafile), טכנולוגיית ההתקן תהיה זו של ההתקן שהתייחסנו אליו קודם כשנמסר לפונקציה CreatEnhMetaFile. כדי לקבוע אם ההקשר הוא הקשר התקן של קובץ-על משופר, צריך להשתמש בפונקציה GetObjectType.

לבסוף, אם התוכנית חייבת לזהות את תכונות הרשת של ההתקן שדנים בו, הקריאה ל-GetDeviceCaps תכלול את הקבוע RASTERCAPS. כאשר התוכנית דורשת את תכונות הרשת של התקן, הערך המוחזר של הפונקציה הוא ערך אחד או יותר מאלה המפורטים בטבלה 6.7.

טבלה 6.7: הערכים האפשריים המוחזרים עבור הדרישה **RASTERCAPS**.

יכולת	פירוש
RC_BANDING	דורש תמיכת פס (Banding Support)
RC_BITBLT	יכול להעביר מפות סיביות
RC_BITMAP64	יכול לתמוך במפות סיביות שגדולות מ- 64K
RC_DI_BITMAP	יכול לתמוך בפונקציות SetDIBits ו-GetDIBits
RC_DIBTODEV	יכול לתמוך בפונקציה SetDIBitsToDevice
RC_FLOODFILL	יכול לבצע מילוי צבע לפי אלגוריתם FloodFill
RC_GDI20_OUTPUT	יכול לתמוך בעצמים של Windows 9x
RC_PALETTE	מגדיר לוח צבעים מבוסס התקן
RC_SCALING	יכול לדרג (Capable Of Scaling)
RC_STRETCHBLT	יכול להפעיל את הפונקציה StretchBlt
RC_STRETCHDIB	יכול להפעיל את הפונקציה StretchDIBits

בנוסף למידע הדרוש לתוכניות על יכולת הרישות של ההתקן (Rastering Capabilities), הן צריכות לעיתים קרובות מידע אודות יכולת ההתקן לצייר ולהפיק פלט בצורת עקומות (Curved Figures). באפשרותך לבדוק את יכולת ציור העקומות בעת הצגת הדרישה CURVECAPS. הדרישה CURVECAPS מחזירה ערך אחד או יותר מהערכים שמפורטים בטבלה 6.8.

טבלה 6.8: הערכים המוחזרים כתוצאה מהקריאה ל-**CURVECAPS**.

ערך	פירוש
CC_NONE	ההתקן אינו תומך בעקומות
CC_CIRCLES	ההתקן יכול לצייר מעגלים
CC_PIE	ההתקן יכול לצייר פרוסות עוגה (Pie Wedges)
CC_CHORD	ההתקן יכול לצייר מקטע אליפטי (Chord Arc)
CC_ELLIPSES	ההתקן יכול לצייר אליפסות
CC_WIDE	ההתקן יכול לצייר גבולות רחבים (Wide Borders)
CC_STYLED	ההתקן יכול לצייר גבולות עם סגנון (Styled Borders)
CC_WIDESTYLED	ההתקן יכול לצייר גבולות רחבים ועם סגנון
CC_INTERIORS	ההתקן יכול לצייר פנים (Interior)
CC_ROUNDRECT	ההתקן יכול לצייר מלבנים מעוגלים (Rounded Rectangles)

להתקן יכולות להיות אפשרויות ציור של עקומות, או שאינו יכול לעשות זאת. למרבית ההתקנים יש גם יכולות ציור קווים מסוימות. באפשרות התוכניות לבדוק את יכולות הציור של קווים עם הקריאה LINECAPS. טבלה 6.9 מפרטת את הערכים האפשריים המוחזרים כתוצאה מהדרישה LINECAPS.

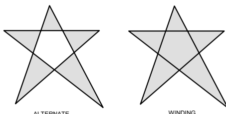
טבלה 6.9: הערכים המוחזרים כתוצאה מהקריאה **LINECAPS**.

ערך	פירוש
LC_NONE	ההתקן אינו תומך בציור קווים
LC_POLYLINE	ההתקן יכול לצייר מצולע (Polyline)
LC_MARKER	ההתקן יכול לצייר סימן (Marker)
LC_POLYMARKER	ההתקן יכול לצייר סימנים רבים (Multiple Markers)
LC_WIDE	ההתקן יכול לצייר קווים רחבים
LC_STYLED	ההתקן יכול לצייר קווים עם סגנון
LC_WIDESTYLED	ההתקן יכול לצייר קווים רחבים ועם סגנון
LC_INTERIORS	ההתקן יכול לצייר פנים (Interior)

בנוסף למידע על יכולת ההתקן לצייר קווים ומעגלים, התוכנית דורשת במקרים רבים לדעת על יכולת ההתקן לצייר מצולעים (Polygons) ובהם מלבנים, טרפזים וכדומה. באפשרותך לבדוק את יכולות ההתקן לעשות זאת, על ידי שימוש ב-POLYGONALCAPS, אשר מחזירה ערך אחד או יותר מהערכים שמפורטים בטבלה 6.10.

טבלה 6.10: הערכים האפשריים המוחזרים כתוצאה מהקריאה ל-**POLYGONALCAPS**.

ערך	פירוש
PC_NONE	ההתקן אינו תומך במצולעים
PC_POLYGON	ההתקן יכול לצייר מצולעים ממולאים חילופית (Alternate-Fill Polygons). כלומר, Windows ממלאת רק את שטחי הפנים שאפשר להגיע אליהם מחוץ לצורה על ידי חציית מספר אי-זוגי של גבולות. התבונן בתרשים 6.2.
PC_RECTANGLE	ההתקן יכול לצייר מלבנים.
PC_WINDPOLYGON	ההתקן יכול לצייר מצולעים שכל שטחי הפנים שלהם ממולאים (Winding-Fill Polygons). התבונן בתרשים 6.2.
PC_SCANLINE	ההתקן יכול לצייר קוו יחיד מדורג.
PC_WIDE	ההתקן יכול לצייר גבולות רחבים.
PC_STYLED	ההתקן יכול לצייר גבולות עם סגנון.
PC_WIDESTYLED	ההתקן יכול לצייר גבולות רחבים ועם סגנון.
PC_INTERIORS	ההתקן יכול לצייר פנים (Interior).



תרשים 6.2: מילוי הפוליגון לפי Alternate ולפי Winding.

לבסוף, פעמים רבות התוכנית דורשת מידע על יכולות תצוגת הטקסט של ההתקן. באפשרות התוכניות לבדוק את יכולות תצוגת הטקסט של ההתקן על ידי קריאה לפונקציה DeviceCaps עם הפרמטר TEXTCAPS. פרמטר זה מייצג ערך המציין את יכולות תצוגת הטקסט. ראה טבלה 6.11.

טבלה 6.11: הערכים האפשריים המוחזרים כתוצאה מהקריאה ל-**TEXTCAPS**.

ערך	פירוש
TC_OP_CHARACTER	יש להתקן יכולת דיוק של פלט תווי (Character Output Precision).
TC_OP_STROKE	יש להתקן יכולת דיוק של פלט תווי (Stroke Output Precision).
TC_CP_STROKE	יש להתקן יכולת דיוק של גזירה תווית (Stroke Clip Precision).
TC_CR_90	ההתקן יכול לסובב את התו ב- 90 מעלות.
TC_CR_ANY	ההתקן יכול לסובב כל תו.
TC_SF_X_YINDEP	ההתקן יכול לדרג בצורה נפרדת לאורך ציר x ולאורך ציר y.
TC_SA_DOUBLE	להתקן יכולת דירוג של תווים כפולים.
TC_SA_INTEGER	ההתקן משתמש בכפולות של מספרים שלמים בלבד לדירוג תווים.
TC_SA_CONTIN	ההתקן משתמש בכל כפולה שהיא לדירוג מדויק של התו.
TC_EA_DOUBLE	ההתקן יכול לצייר תווים בעלי משקל כפול (הדגשה/עובי).
TC_IA_ABLE	ההתקן יכול להטות תווים.
TC_UA_ABLE	ההתקן יכול לשלב קו תחתון.
TC_SO_ABLE	ההתקן יכול לצייר קווים חוצים.
TC_RA_ABLE	ההתקן יכול לצייר גופני רשת (Raster Fonts).
TC_VA_ABLE	ההתקן יכול לצייר גופני ווקטור (Vector Fonts).
TC_RESERVED	שמור; חייב להיות אפס.
TC_SCROLLBLT	ההתקן אינו יכול לגלול על ידי שימוש בהעברת בלוק סיביות (Bit-Block Transfer).

כדי להבין יותר טוב את פעולת הפונקציה GetDeviceCaps, התבונן בתוכנית **Get_DevC**, שבתקליטור המצורף לספר זה (Chap06). התוכנית משתמשת בפונקציה CreateIC כדי ליצור הקשר מידע למסך. התוכנית **Get_DevC** משתמשת בהקשר שנוצר, כדי למצוא מספר הסיביות לפיקסל ומספר מישורי הצבע עבור הצג. התוכנית **Get_DevC** משתמשת ב-GetDeviceCaps כדי לעשות כל החלטה. הקוד שמבצע את העיבוד המעשי, נמצא כמו תמיד בפונקציה WndProc.

6.10 הפונקציה GetSystemMetrics לניתוח חלון

התוכניות יכולה לנהל כמעט כל מאפיין שיש לחלון, בין אם הוא חלון-אב, חלון-בן, או חלון פקד. אחת מפעולות הניהול המקובלת והנפוצה ביותר שהתוכניות חייבות לבצע היא עדכון גודל התצוגה הנוכחית, כדי שתתאים לגודל היחסי של צג המחשב. אפשר להשיג מידע זה, בנוסף למידע אחר על ההגדרות שנמצאות במערכת המחשב שלך, על ידי שימוש בפונקציה GetSystemMetrics. פונקציה זו מקבלת קשת רחבה של **מידות מערכת** (System Metrics) שונות ו**תצורת מערכת** (System Configuration). מידות מערכת הן המימדים (רוחב וגובה) של האלמנטים שמוצגים על ידי Windows. כל המימדים שמקבלת הפונקציה GetSystemMetrics הם בפיקסלים. את הפונקציה GetSystemMetrics כותבים כמו בהגדרה שלהלן:

```
int GetSystemMetrics(int nIndex);
```

הפרמטר nIndex מגדיר את מידת המערכת או את הגדרת התצורה ש-GetSystemMetrics צריכה לקבל. כל ערכי *SM_CX הם מידות רוחב. כל ערכי *SM_CY הם מידות גובה. Windows API מגדיר את הערכים המפורטים בטבלה 6.12.

טבלה 6.12: מידות המערכת שניתן לאחזר על ידי הפונקציה GetSystemMetrics.

ערך	פירוש
SM_ARRANGE	דגלים שמגדירים איך המערכת מסדרת חלונות ממוזערים (כלומר מוקטנים לסמל).
SM_CLEANBOOT	ערך אשר מגדיר איך המשתמש איתחל את המערכת. שלושת הערכים האפשריים הם 0 - **אתחול רגיל** (normal boot), 1 - אתחול **אל-כשל** (Fail-safe), ו- 2 - אתחול **אל-כשל עם אתחול רשת** (Fail-Safe with Network Boot). אתחול אל-כשל (נקרא גם **אתחול בטוח**, SafeBoot) עוקף את קבצי האתחול של המשתמש.
SM_CMOUSEBUTTONS	מספר הלחצנים בעכבר; או אפס, אם לא מותקן עכבר.
SM_CXBORDER	הרוחב של גבול החלון, בפיקסלים. ערך זה שקול לערך SM_CXEDGE לחלונות שיש להם מראה תלת מימדי (3-D look).
SM_CYBORDER	הגובה של גבול החלון, בפיקסלים. ערך זה שקול לערך SM_CYEDGE לחלונות שיש להם מראה תלת מימדי (3-D look).

ערך	פירוש
SM_CXCURSOR	רוחב הסמן, בפיקסלים. אלה הם מימדי הסמן שנתמכים על ידי מנהל התצוגה הנוכחי. המערכת אינה יכולה ליצור סמנים בגדלים אחרים.
SM_CYCURSOR	גובה הסמן, בפיקסלים. אלה הם מימדי הסמן שנתמכים על ידי מנהל התצוגה הנוכחי. המערכת אינה יכולה ליצור סמנים בגדלים אחרים.
SM_CXDLGFRAME	כמו SM_CXFIXEDFRAME.
SM_CYDLGFRAME	כמו SM_CYFIXEDFRAME.
SM_CXDOUBLECLK	רוחב בפיקסלים, של המלבן סביב מיקום הלחיצה הראשונה בסדרת לחיצה כפולה. הלחיצה השנייה חייבת להתרחש בתוך מלבן זה, כדי שהמערכת תקבל את שתי הלחיצות כלחיצה כפולה (שתי הלחיצות חייבות גם כן להתרחש בפרק זמן מוגדר).
SM_CYDOUBLECLK	גובה בפיקסלים, של המלבן סביב מיקום הלחיצה הראשונה בסדרת לחיצה כפולה. הלחיצה השנייה חייבת להתרחש בתוך מלבן זה כדי שהמערכת תקבל את שתי הלחיצות כלחיצה כפולה (שתי הלחיצות חייבות גם כן להתרחש בפרק זמן מוגדר).
SM_CXDRAG	רוחב בפיקסלים, של מלבן אשר ממורכז סביב נקודת הגרירה, כדי להרשות תנועה מוגבלת למצביע העכבר לפני שמתחילה פעולת גרירה. דבר זה מאפשר למשתמש ללחוץ ולשחרר את לחצן העכבר בקלות, מבלי להתחיל בפעולת גרירה.
SM_CYDRAG	גובה בפיקסלים, של מלבן אשר ממורכז סביב נקודת הגרירה אשר מאפשרת למשתמש להזיז את מצביע העכבר למרחק מוגבל לפני שמתחילה פעולת גרירה. מלבן כזה מאפשר למשתמש ללחוץ ולשחרר את לחצן העכבר בקלות מבלי להתחיל בפעולת גרירה.
SM_CXEDGE	רוחב בפיקסלים, של גבול תלת-מימדי. SM_CXEDGE הוא המקביל התלת-מימדי של SM_CXBORDER.
SM_CYEDGE	גובה, בפיקסלים, של גבול תלת-ממדי. SM_CXEDGE הוא המקביל התלת-מימדי של SM_CYBORDER.

ערך	פירוש
SM_CXFIXEDFRAME	עובי בפיקסלים, של המסגרת סביב היקף החלון שיש לו כותרת (Caption), אך אי אפשר לשנות את גודלו. SM_CXFIXEDFRAME הוא רוחב הגבול האופקי. כמו SM_CXDLGFRAME.
SM_CYFIXEDFRAME	עובי בפיקסלים, של המסגרת סביב היקף החלון שיש לו כותרת (Caption), אך אי אפשר לשנות את גודלו. SM_CYFIXEDFRAME הוא גובה הגבול האנכי. כמו SM_CYDLGFRAME.
SM_CXFRAME	כמו SM_CXSIZEFRAME.
SM_CYFRAME	כמו SM_CYSIZEFRAME.
SM_CXFULLSCREEN	רוחב שטח הלקוח עבור חלון במסך מלא. כדי לקבל את הקואורדינטות של חלק המסך שהסריקה אינה מכסה, קרא לפונקציה SystemParametersInfo עם הערך שמחזירה GetSystemMetrics.
SM_CYFULLSCREEN	גובה שטח הלקוח עבור חלון במסך מלא. כדי לקבל את הקואורדינטות של חלק המסך שהסריקה אינה מכסה אותו, קרא לפונקציה SystemParametersInfo עם הערך SPI_GETWORKAREA.
SM_CXHSCROLL	רוחב בפיקסלים, של מפת הסיביות של החץ שבפס הגלילה האופקי.
SM_CYHSCROLL	גובה, בפיקסלים, של פס גלילה אופקי.
SM_CXHTHUMB	רוחב בפיקסלים, של תיבת הגררה בפס גלילה אופקי.
SM_CXICON	רוחב ברירת המחדל של סמל, בפיקסלים. ערך זה הוא בדרך כלל 32x32, אבל יכול להשתנות כתלות בתצוגת החומרה המותקנת. הפונקציה LoadIcon יכולה לטעון רק סמלים במימדים אלה.
SM_CYICON	גובה ברירת המחדל של סמל, בפיקסלים. ערך זה הוא בדרך כלל 32x32, אבל יכול להשתנות כתלות בתצוגת החומרה המותקנת. הפונקציה LoadIcon יכולה לטעון רק סמלים במימדים אלה.

ערך	פירוש
SM_CXICONSPACING	מימד X, בפיקסלים, של תאי שריג (Grid) עבור פריטים בתצוגת סמלים גדולים. כל פריט מתאים למלבן בגודל זה כאשר מסדרים אותם. ערכים אלה תמיד גדולים מ-SM_CXICON ומ-SM_CYICON, או שווים להם.
SM_CYICONSPACING	מימד Y, בפיקסלים, של תאי שריג עבור פריטים בתצוגת סמלים גדולים. כל פריט מתאים למלבן בגודל זה כאשר מסדרים אותם. ערכים אלה תמיד גדולים מ-SM_CXICON ומ-SM_CYICON, או שווים להם.
SM_CXMAXIMIZED	ערך ברירת המחדל של מימד X, בפיקסלים, של חלון בגודל מקסימלי שאין לו אב.
SM_CYMAXIMIZED	ערך ברירת המחדל של מימד Y, בפיקסלים, של חלון בגודל מקסימלי שאין לו אב.
SM_CXMAXTRACK	ערך ברירת המחדל לרוחב המקסימלי, בפיקסלים, של חלון שיש לו כותרת וגבולות שאפשר לשנות את גודלם. המשתמש אינו יכול לגרור את מסגרת החלון לגודל שמעבר לרוחב זה. החלון יכול לטפל בהודעה WM_GETMINMAXINFO כדי לעקוף ערכים אלה.
SM_CYMAXTRACK	ערך ברירת המחדל לגובה המקסימלי, בפיקסלים, לחלון שיש לו כותרת וגבולות שאפשר לשנות את גודלם. המשתמש אינו יכול לגרור את מסגרת החלון לגודל שמעבר לגובה זה. החלון יכול לטפל בהודעה WM_GETMINMAXINFO כדי לעקוף ערכים אלה.
SM_CXMENUCHECK	רוחב, בפיקסלים, של הגדרת ברירת המחדל למפת הסיביות שמשמשת לסימון פריט בתפריט.
SM_CYMENUCHECK	גובה, בפיקסלים, של הגדרת ברירת המחדל למפת הסיביות שמשמשת לסימון פריט בתפריט.
SM_CXMENUSIZE	רוחב, בפיקסלים, של לחצני שורת התפריט, כמו סגירת בן במסמכים מרובים (MDI).
SM_CYMENUSIZE	גובה, בפיקסלים, של לחצני שורת התפריט, כמו סגירת בן במסמכים מרובים (MDI).
SM_CXMIN	רוחב מינימלי, בפיקסלים, של חלון.

ערך	פירוש
SM_CYMIN	גובה מינימלי, בפיקסלים, של חלון.
SM_CXMINIMIZED	רוחב, בפיקסלים, של חלון נורמלי מוקטן לסמל (Minimized).
SM_CYMINIMIZED	גובה, בפיקסלים, של חלון נורמלי מוקטן לסמל (Minimized).
SM_CXMINSPACING	רוחב, בפיקסלים, של **תא שריג** (Grid Cell) עבור חלון מוקטן לסמל. כל החלונות המוקטנים לסמל (ממוזערים) מתאימים למלבן בגודל זה כשמסדרים אותם. ערכים אלה תמיד גדולים או שווים ל-SM_CXMINIMIZED ול-SM_CYMINIMIZED.
SM_CYMINSPACING	גובה, בפיקסלים, של **תא שריג** (Grid Cell) עבור חלון מוקטן לסמל. כלהחלונות המוקטנים לסמל (ממוזערים) מתאימים למלבן בגודל זה כשמסדרים אותם. ערכים אלה תמיד גדולים או שווים ל-SM_CXMINIMIZED ול-SM_CYMINIMIZED.
SM_CXMINTRACK	רוחב עקיבה מינימלי של חלון, בפיקסלים. המשתמש אינו יכול לגרור את מסגרת החלון לגודל שקטן מערך זה. החלון יכול לטפל בהודעה WM_GETMINMAXINFO כדי לעקוף ערכים אלה.
SM_CYMINTRACK	גובה עקיבה מינימלי של חלון, בפיקסלים. המשתמש אינו יכול לגרור את מסגרת החלון לגודל שקטן מערך זה. החלון יכול לטפל בהודעה WM_GETMINMAXINFO כדי לעקוף ערכים אלה.
SM_CXSCREEN	רוחב המסך, בפיקסלים.
SM_CYSCREEN	גובה המסך, בפיקסלים.
SM_CXSIZE	רוחב, בפיקסלים, של לחצן ב**כותרת החלון** (Wondow's Caption), או ב**שורת הכותרת** (Title Bar).
SM_CYSIZE	גובה, בפיקסלים, של לחצן ב**כותרת החלון** (Wondow's Caption), או ב**שורת הכותרת** (Title Bar).
SM_CXSIZEFRAME	עובי, בפיקסלים, של הגבול שאפשר לשנות את גודלו מסביב להיקף החלון שהמשתמש יכול לשנות את גודלו. SM_CXSIZEFRAME הוא הרוחב של הגבול האופקי. כמו SM_CXFRAME.

ערך	פירוש
SM_CYSIZEFRAME	עובי, בפיקסלים, של הגבול שאפשר לשנות את גודלו מסביב להיקף החלון שהמשתמש יכול לשנות את גודלו. SM_CYSIZEFRAME הוא הגובה של הגבול האנכי. כמו SM_CYRAME.
SM_CXSMICON	רוחב מומלץ של הסמל הקטן, בפיקסלים. סמלים קטנים מופיעים בדרך כלל בכותרת החלון ובתצוגת סמל קטן.
SM_CYSMICON	גובה מומלץ של הסמל הקטן, בפיקסלים. סמלים קטנים מופיעים בדרך כלל בכותרת החלון ובתצוגת סמל קטן.
SM_CXSMSIZE	רוחב של לחצני כיתוב קטנים, בפיקסלים.
SM_CYSMSIZE	גובה של לחצני כיתוב קטנים, בפיקסלים.
SM_CXVSCROLL	רוחב של מפת סיביות החץ בפס גלילה אנכי, בפיקסלים.
SM_CYVSCROLL	גובה של מפת סיביות החץ בפס גלילה אנכי, בפיקסלים.
SM_CYCAPTION	גובה האזור הנורמלי לכיתוב, בפיקסלים.
SM_CYCANJIWINDOW	עבור גרסאות Windows התומכות במערך תווים של בתים כפולים (DBCS, או Double-Byte Character Set), זהו הגובה של החלון בתחתית המסך, בפיקסלים.
SM_CYMENU	גובה, בפיקסלים, של שורת תפריט אחת.
SM_CYSMCAPTION	גובה של **כותרת קטנה** (Small Caption) , בפיקסלים.
SM_CYVTHUMB	גובה של **תיבת הגררה** (Thumb Box) בפס גלילה אנכי, בפיקסלים.
SM_DBCSENABLED	ערך True או שאינו אפס, אם מותקנת גרסת מערך התווים של בתים כפולים (DBCS) של USER.EXE ; אחרת - False, או אפס.
SM_DEBUG	ערך True או שאינו אפס, אם מותקנת גרסת **מנפה השגיאות** (Debugging Version) של USER.EXE ; אחרת - False, או אפס.

ערך	פירוש
SM_MENUDROPALIGNMENT	ערך True או שאינו אפס, אם התפריטים הנשלפים כלפי מטה (Drop-Down Menus) מיושרים ימינה, יחסית לפריט התפריט המתאים; ואם הם מיושרים שמאלה - False, או אפס.
SM_MIDEASTENABLED	ערך True אם המערכת מותאמת לשפות עברית/ערבית.
SM_MOUSEPRESENT	ערך True או שאינו אפס, אם מותקן עכבר; אחרת - False, או אפס.
SM_MOUSEWHEELPRESENT	רק תחת Windows NT, ערך True או שאינו אפס, אם מותקן עכבר עם גלגל; אחרת - False, או אפס.
SM_NETWORK	הפונקציה מחזירה ערך, שבו הסיבית הפחות משמעותית נקבעת אם עובדים ברשת; אחרת, הפונקציה מחזירה ערך שבו הסיבית הפחות משמעותית אינה נקבעת (כבויה או נקייה). Windows שומרת את שאר הסיביות לשימוש עתידי.
SM_PENWINDOWS	ערך True או שאינו אפס, אם מותקנות ההרחבות של מחשב העט של Windows; אחרת - False, או אפס.
SM_SECURE	ערך True אם תכונת האבטחה מופעלת; אחרת - False.
SM_SHOWSOUNDS	ערך True או שאינו אפס, אם המשתמש מבקש מהיישום להציג מידע בצורה ויזואלית במקרים שהמידע היה מוצג בפורמט קולי (Audible Form); אחרת - False, או אפס.
SM_SLOWMACHINE	ערך True אם יש למחשב מעבד איטי (Low-End); אחרת - False.
SM_SWAPBUTTON	ערך True או שאינו אפס, אם הפונקציונליות של לחצני העכבר מוחלפת; אחרת - False, או אפס.

אם הפונקציה מצליחה, הערך המוחזר הוא הערך של המערכת או הגדרת התצורה שביקשנו לדעת; אם הפונקציה נכשלת, הערך המוחזר הוא אפס.

יכול להיות שמידות המערכת ישתנו מתצוגה אחת לאחרת. הקבוע SM_ARRANGE מגדיר איך המערכת מסדרת חלונות מוקטנים לסמל, ומורכב ממקום התחלה וכיוון. מקום ההתחלה יכול להיות אחד הערכים שמפורטים בטבלה 6.13.

טבלה 6.13: ערכי נקודות ההתחלה.

ערך	פירוש
ARW_BOTTOMLEFT	מתחיל מפינה שמאלית תחתונה במסך (ברירת המחדל).
ARW_BOTTOMRIGHT	מתחיל מפינה ימנית תחתונה במסך. שקול ל-ARW_STARTRIGHT.
ARW_HIDE	מסתיר חלונות מוקטנים לסמל, על ידי העברתם מהשטח הנראה של המסך.
ARW_TOPLEFT	מתחיל מפינה שמאלית עליונה במסך. שקול ל-ARV_STARTTOP.
ARW_TOPRIGHT	מתחיל מפינה ימנית עליונה במסך. שקול ל-ARW_STARTTOP | SRW_STARTRIGHT.

הכיוון אשר המערכת מסדרת בו חלונות מוקטנים לסמל, יכול להיות אחד מהערכים שמפורטים בטבלה 6.14.

טבלה 6.14: ערכי הסדר האפשריים של סמלי חלונות.

ערך	פירוש
ARW_DOWN	סדר אנכי, מלמעלה למטה
ARW_LEFT	סדר אופקי, משמאל לימין
ARW_RIGHT	סדר אופקי, מימין לשמאל
ARW_UP	סדר אנכי, מלמטה למעלה

הסעיף הבא מפרט איך התוכניות יכולות להשתמש במידות מערכת (System Metrics) בזמן העיבוד שלהן.

6.11 הפונקציה GetSystemMetrics

בסעיף קודם למדת שבאפשרות התוכניות להשתמש בפונקציה GetSystemMetrics כדי להשיג מידע על המידות הנוכחיות של Windows בכל מחשב נתון. ככל שהתוכניות הופכות להיות יותר ויותר מורכבות, וככל שתבנה יישומים שיופעלו על מחשבים שונים, בדיקות מידות המערכת הופכת להיות חשובה לפני כל קביעת תצוגת מסכים, כי הדבר יכול להבטיח שתצוגת המסך תתאים גם למסכי 640x480 וגם למסכי 1024x768. לדוגמה, התוכנית **Get_Sysm**, שבתקליטור המצורף לספר זה (chap06), משתמשת ב-GetSystemMetrics כדי לבחון את המצב הנוכחי של גודל החלון, גודל המסך ומידע אחר. כאשר המשתמש בוחר באפשרות !Test מהתפריט, התוכנית מציגה את המידע בחלון.

6.12 קבלת הקשר התקן עבור החלון כולו

התוכניות יפעלו בדרך כלל עם הקשר התקן אל שטח הלקוח של החלון, אך לפעמים נדרש הקשר התקן עבור החלון כולו. כדי להשיג הקשר התקן לחלון כולו, באפשרות התוכנית להשתמש בפונקציה GetWindowDC. פונקציה זו מקבלת את הקשר ההתקן (DC) של החלון בשלמותו, כולל שורת כותרת, תפריטים ופסי גלילה. הקשר התקן לחלון מאפשר ציור בכל מקום בחלון, מפני שנקודת המוצא של הקשר ההתקן היא הפינה השמאלית העליונה של החלון, ולא הפינה השמאלית העליונה של שטח הלקוח.

GetWindowDC מציבה מאפייני ברירת המחדל אל הקשר ההתקן של החלון בכל פעם שהיא מקבלת את הקשר ההתקן. מכיון ש-GetWindowDC מציבה מאפייני ברירת מחדל להקשר ההתקן של החלון, מאבדים את המאפיינים שהיו קודם. את הפונקציה GetWindowDC כותבים כמו בהגדרה שלהלן:

```
HDC GetWindowDC(HWND hWnd);
```

הפרמטר hWnd מזהה את החלון עם הקשר התקן אשר GetWindowDC עומדת לקבל. אם הפונקציה מצליחה, הערך המוחזר הוא ידית של הקשר התקן לחלון המוגדר; אם הפונקציה נכשלת, הערך המוחזר הוא NULL, שמציין שגיאה או פרמטר hWnd שאינו תקף.

השימוש ב-GetWindowDC מיועד לספק אפקטי צביעה מיוחדים בשטח החלון שאינו שטח הלקוח. הצביעה בכל שטח שאינו שטח הלקוח בחלון אינה מומלצת. באפשרותך להשתמש בפונקציה GetSystemMetrics כדי לקבל את המימדים לחלקים שונים של השטח שאינו שטח הלקוח, כמו שורת הכותרת, תפריט ופסי גלילה. לאחר סיום הצביעה, התוכנית חייבת לקרוא לפונקציה ReleaseDC כדי לשחרר את הקשר ההתקן. אי שחרור הקשר ההתקן של החלון יכול להשפיע באופן חמור על בקשות צביעה עוקבות של היישום.

כדי להבין יותר טוב את פעולת הפונקציה GetWindowDC, התבונן בתוכנית **GetWinDC**, שבתקליטור המצורף לספר זה (בתיקיה Chap06). התוכנית משתמשת בפונקציה GetWindowDC כדי לקבל הקשר התקן עבור החלון כולו. אחר כך היא משתמשת ב-GetSystemMetrics כדי לקבוע את מידות הגבולות והכותרת. לבסוף, התוכנית משתמשת בהקשר ההתקן של החלון כולו כדי לצבוע תבנית בשורת הכותרת של החלון. הקוד המעשי מתרחש בתוך הפונקציה WndProc.

6.13 שחרור הקשר התקן

שחרור אובייקט לאחר שהתוכנית מסיימת את הטיפול שלה באובייקט, הוא חלק חשוב בתכנות בשפת ++C. ניהול הקשרי התקן אינו יוצא מן הכלל. הפונקציה ReleaseDC משחררת הקשר התקן (DC), כך שיישומים אחרים יכולים להשתמש בו. פעולת הפונקציה ReleaseDC תלויה בסוג הקשר ההתקן. היא משחררת רק הקשרי התקן משותפים והקשרי התקן לחלונות. אין לה שום השפעה על הקשרי התקן מסוג class (מחלקה) או private (פרטי). את הפונקציה ReleaseDC כותבים בתוכנית כמו בהגדרה שלהלן:

```
int ReleaseDC(HWND hWnd, HDC hDC);
```

הפרמטר hWnd מזהה את החלון אשר הפונקציה ReleaseDC עומדת לשחרר את הקשר ההתקן שלו. הפרמטר hDC מזהה את הקשר ההתקן אשר הפונקציה ReleaseDC עומדת לשחרר. הערך המוחזר מגדיר אם ReleaseDC הצליחה לשחרר את הקשר ההתקן. אם ReleaseDC מצליחה לשחרר את הקשר ההתקן, היא מחזירה 1; אם ReleaseDC אינה מצליחה לשחרר את הקשר ההתקן, היא מחזירה אפס.

היישום חייב לקרוא לפונקציה ReleaseDC עבור כל קריאה לפונקציה GetWindowDC ועבור כל קריאה לפונקציה GetDC אשר מקבלת הקשר התקן משותף. היישום אינו יכול להשתמש בפונקציה ReleaseDC כדי לשחרר הקשר התקן אשר נוצר באמצעות הפונקציה CreateDC. במקרה האחרון, היישום חייב להשתמש בפונקציה DeleteDC.

6.14 קבלת ידית חלון מהקשר ההתקן

התוכניות עשויות לבצע פעולות כלליות בכל הקשר התקן נתון. אך לעיתים ברצונך להימנע מביצוע עיבוד כלשהו בהקשר התקן שקשור עם חלון מסוים. עם הפונקציה WindowFromDC, יכולה התוכנית להמיר את הקשר ההתקן לידית חלון. הפונקציה WindowFromDC מחזירה את ידית החלון שקשור עם הקשר התקן התצוגה הנתון. פונקציות פלט אשר משתמשות בהקשר ההתקן מציירות בחלון אשר הידית שלו הוחזרה על ידי WindowFromDC. את הפונקציה WindowFromDC כותבים בתוכנית כמו בהגדרה שלהלן:

```
HWND WindowFromDC(HDC hDC);
```

הפרמטר hDC מזהה את הקשר ההתקן אשר הפונקציה WindowFromDC עומדת לקבל הידית שלו עבור החלון שקשור להקשר ההתקן. כאשר הפונקציה מצליחה, הערך המוחזר הוא ידית לחלון שקשור להקשר ההתקן לתצוגה; אם הפונקציה נכשלת, הערך המוחזר הוא NULL.

☜ אזהרה: סיכוני השימוש ב-CreateDC

למדת שהתוכניות ישתמשו לעיתים תכופות בפונקציה CreateDC כדי להשיג הקשר התקן עבור המדפסת. עם זאת, התוכניות יכולות להשתמש גם ב-CreateDC כדי להשיג הקשר התקן אל מסך הצג (מסך החומרה, לא שטח לקוח של חלון בודד, או אפילו שטח שולחן העבודה). כאשר משתמשים ב-CreateDC כדי להשיג הקשר התקן אל המסך, התוכניות יכולות למעשה לצייר בכל מקום על פני המסך, ולא רק בתוך גבולות שטח התוכנית. בנוסף לתוצאות הפוטנציאליות שאי אפשר לצפות להן, טיפול כזה אינו מתאים לתקן של Windows. כאשר מנסים להשיג הקשר התקן אל חלון במסך, על התוכניות שלך להשתמש תמיד בפונקציה GetDC, או בפונקציה BeginPaint, ולא בפונקציה CreateDC.

פרק **7**

מפות סיביות, קבצי-על וסמלים (Bitmaps, Metafiles ו-Icons)

7.1 מפות סיביות שתלויות בהתקן

מפות סיביות הן בלוקים של נתונים אשר התוכניות יכולות להוציא ישירות כפלט אל התקן, כמו צג למשל. אפשר לחשוב על מפות סיביות כדרך לאחסון המידע של הפיקסל ישירות מהמסך אל מאגר זיכרון. צביעת מפות סיביות על המסך היא יותר מהירה מהשימוש בפונקציות ממשק ההתקן הגרפי כמו Rectangle ו-LineTo. החיסרון של מפות סיביות הוא בכך שהן צריכות כמות גדולה של זיכרון ומקום בדיסק, ואי אפשר לדרג אותן (Scale) בצורה טובה, במיוחד אם הן מכילות טקסט. כאשר מדרגים מפת סיביות, היא מאבדת מהאיכות, וגורמת לעיוות הטקסט.

Windows מספקת שתי סוגים של מפות סיביות: **מפות סיביות שתלויות התקן** (Device-Dependent Bitmaps) **ומפות סיביות שאינן תלויות התקן** (Device-Independent Bitmaps). מפות סיביות תלויות התקן הן תבנית ישנה, וכפי שמשתמע מהשם שלהן, הן פחות גמישות מאשר מפות סיביות שאינן תלויות התקן. כל היישומים של Win32 שתכתוב צריכות להשתמש במפות סיביות שאינן תלויות התקן. עם זאת, Windows מספקת את מרבית פונקציות Win32 שעוסקות במפות סיביות תלויות התקן, כדי לאפשר ליישומי Windows קודמים, בעלי 16 סיביות, לפעול.

ככלל, יוצרים מפות סיביות באמצעות תוכנית ציור כמו **הצייר** של מיקרוסופט (Microsoft Paint), או עם עורך מפות סיביות, שנמצא בדרך כלל כמרכיב בסביבת פיתוח מוכללת (כמו Borland C++ 5.02, או Visual C++ 5.0). אחר כך מאחסנים את מפת הסיביות בדיסק עם סיומת שם קובץ BMP. Windows שומרת את מפת הסיביות כמפת סיביות שאינה תלויית התקן, וממירה אותה למפת סיביות תלויית התקן כאשר התוכנית קוראות ל-LoadBitmap. אפשר להציב את מילת המפתח BITMAP לפני שם קובץ מפת הסיביות, כדי שתוכל להוסיף מפות סיביות לקובץ המשאבים, כפי שמוצג בדוגמה הבאה:

```
pen BITMAP pen.bmp
```

כדי להבין יותר טוב את התוכנית לניהול מפות סיביות, התבונן בתוכנית **Pen_BMP**, שבתקליטור המצורף לספר זה (בתיקיה Chap07). התוכנית טוענת את מפת הסיביות Pen, ממקמת אותה בהקשר התקן שבזיכרון, ואחר כך מציירת אותה בהקשר התקן התצוגה. קטע הקוד שלהלן מהקובץ **Pen_BMP** מציג את חלק העיבוד המעשי של התוכנית:

```
HBITMAP hBitmap;
HDC                             hDC;
HDC                             hMemDC;

//  Load the bitmap into memory

//hBitmap = LoadBitmap( hInst, "pen" );
hBitmap = LoadBitmap( hInst, "BITMAP1" );

//  Paint the bitmap onto the MemDC and then the screen

hDC = GetDC( hWnd );
hMemDC = CreateCompatibleDC( hDC);
SelectObject( hMemDC, hBitmap );
BitBlt( hDC, 10, 10, 60, 60, hMemDC, 0, 0, SRCCOPY );
DeleteDC( hMemDC );
ReleaseDC( hWnd, hDC );
DeleteObject( hBitmap );
```

תבנית מפת סיביות תלויית התקן פועלת כראוי בהעתקת חלקים של המסך לזיכרון והדבקת חלקים אלה חזרה למקומות אחרים במסך. עם זאת, כאשר היישום חייב לשמור נתונים בקובץ בדיסק ואחר כך להציג אותו בהתקן אחר, תבנית מפת סיביות תלויית התקן אינה טובה עוד. התבנית של מפת סיביות תלויית התקן מניחה שאתה מתכוון תמיד להציג את מפת הסיביות בהתקן שדומה להתקן המקורי שהמפה נוצרה בו, והצבעים בהתקן האחר זהים לאלה שבהתקן המקורי. לרוע המזל, כאשר מציגים את מפת הסיביות בהתקן אחר, הצבעים עלולים להיות שונים. מפת סיביות שאינה תלויית התקן אינה גורמת לבעיות שמתעוררות בשימוש של תבנית מפת סיביות תלויית התקן.

7.2 מפות סיביות שאינן תלויות התקן

לתבנית (פורמט) של מפת סיביות תלויית התקן יש מספר מגבלות, אך תבנית מפת סיביות שאינה תלויית התקן מתגברת על מגבלות אלו. ההבדל המשמעותי ביותר בין מפת סיביות תלויית התקן לבין מפת סיביות שאינה תלויית התקן הוא בכך, שמפת סיביות שאינה תלויית התקן מכילה טבלת צבעים שמפת הסיביות משתמשת בהם.

הכותר (Header) של מפת הסיביות גם כן יותר מורכב בתבנית מפת סיביות שאינה תלויית התקן. מפת סיביות שאינה תלויית התקן, לעומת מפת סיביות תלויית התקן, אינה אובייקט גרפי; כלומר - היא מבנה נתונים. היישום אינו יכול לקבוע מפת סיביות שאינה תלויית התקן אל הקשר ההתקן.

התבנית של מפת סיביות שאינה תלויית התקן מורכבת משלושה חלקי נתונים נפרדים. החלק הראשון של קובץ מפת הסיביות שאינה תלויית התקן הוא המבנה BITMAPINFOHEADER, אשר מוגדר ב- Win32 API כמו שרואים להלן:

```
typedef struct tagBITMAPINFOHEADER {
     DWORD  biSize;
     LONG   biWidth;
     LONG   biHeight;
     WORD   biPlanes;
     WORD   biBitCount;
     DWORD  biCompression;
     DWORD  biSizeImage;
     LONG   biXPelsPerMeter;
     LONG   biYPelsPerMeter;
     DWORD  biClrUsed;
     DWORD  biClrImportant;
} BITMAPINFOHEADER;
```

המבנה BITMAPINFOHEADER מכיל את האיברים שמפורטים בטבלה 7.1.

טבלה 7.1: איברים המבנה BITMAPINFOHEADER.

שם האיבר	תיאור
biSize	מספר הבתים שהמבנה צריך.
BiWidth	רוחב מפת הסיביות, בפיקסלים.
BiHeight	גובה מפת הסיביות, בפיקסלים. אם ערכו של biHeight חיובי, מפת הסיביות אינה תלויית התקן בכיוון למעלה (Bottom-Up), ונקודת המוצא שלה היא הפינה השמאלית התחתונה. אם ערכו של biHeight שלילי, מפת הסיביות אינה תלויית התקן בכיוון למטה (Top-Down) ונקודת המוצא שלה היא הפינה השמאלית העליונה.
BiPlanes	מספר המישורים עבור התקן המטרה. ערך זה חייב להיות 1.
biBitCount	מספר הסיביות לכל פיקסל. ערך זה חייב להיות 1, 4, 8, 16, 24 או 32.

שם האיבר	תיאור
biCompression	סוג הדחיסה עבור מפת סיביות דחוסה מלמטה למעלה (Windows אינה דוחסת מפת סיביות שאינה תלויית התקן מלמעלה למטה). ערך זה יכול להיות אחד מאלה שמפורטים בטבלה 7.2.
biSizeImage	מגדיר את הגודל של התמונה, בבתים. אפשר לקבוע את ערכו של איבר זה לאפס עבור מפות סיביות BI_RGB.
biXPelsPerMeter	הרזולוציה האופקית, בפיקסלים לכל יחידת מידה, של התקן המטרה עבור מפת הסיביות. היישום יכול להשתמש בערך זה כדי לבחור מפת סיביות מקבוצת משאבים אשר תואמת בצורה הטובה ביותר את תכונות ההתקן הנוכחי.
biYPelsPerMeter	הרזולוציה האנכית, בפיקסלים לכל יחידת מידה, של התקן המטרה עבור מפת הסיביות.
BiClrUsed	מספר הצבעים בטבלת הצבעים שמפת הסיביות משתמשת בהם בפועל. אם ערך זה הוא אפס, מפת הסיביות משתמשת במספר הצבעים המקסימלי, בהתאם לערך האיבר biBitCount עבור מצב הדחיסה שמוגדר על ידי biCompression. אם biClrUsed אינו אפס והאיבר biBitCount קטן מ- 16, אז האיבר biClrUsed מגדיר את המספר הממשי של צבעים שמנוע הגרפיקה או מנהל ההתקן ניגשים אליהם. אם biBitCount שווה ל- 16 או יותר גדול, אז האיבר biClrUsed מגדיר את גודל טבלת הצבע שמפת הסיביות שאינה תלויית התקן משתמשת בה, כדי לבצע אופטימיזציה ללוחות הצבעים של Windows. אם biBitCount שווה ל- 16 או 32, לוח הצבעים האופטימלי מתחיל מיד אחרי שלוש המסכות של המילים הכפולות (Three Doubleword Masks). אם מפת הסיביות היא מפת סיביות ארוזה (Packed Bitmap), מפת סיביות שבה מערך מפת הסיביות נמצא מיד אחרי הכותר BITMAPINFO, ואשר מתייחסים אליו על ידי מצביע יחיד), האיבר biClrUsed חייב להיות אפס או הגודל הממשי של טבלת הצבעים.
biClrImportant	מספר אינדקסי הצבעים, אשר הגדרת מפת הסיביות שאינה תלויית התקן מציינת אותם כחשובים לתצוגת מפת הסיביות. אם ערך זה שווה לאפס, כל הצבעים חשובים.

כמו שניתן לראות בטבלה 7.1, אפשר ליצור מפות סיביות מטה-מעלה דחוסות. כאשר התוכנית טוענת מפת סיביות דחוסה, היא צריכה לבדוק את ערך האיבר biCompression כדי לקבוע את סוג הדחיסה. טבלה 7.2 מפרטת את הערכים האפשריים של האיבר biCompression.

טבלה 7.2: הערכים האפשריים עבור האיבר **biCompression**.

ערך	**תיאור**
BI_RGB	פורמט שאינו דחוס.
BI_RLE8	פורמט דחיסה **קידוד רצף** (RLE, Run-Length Encoded)_עבור מפות סיביות עם 8 סיביות לכל פיקסל. פורמט הדחיסה הוא פורמט של שני בתים שמורכבים מבית אחד שמשמש כמונה ואחריו בית שכולל את אינדקס הצבע.
BI_RLE4	פורמט דחיסה **קידוד רצף** (RLE, Run-Length Encoded) עבור מפות סיביות עם 4 סיביות לכל פיקסל. פורמט הדחיסה הוא פורמט של שני בתים שמורכב ממונה בתים ואחריו שתי מילים לאינדקסי צבע.
BI_BITFIELDS	ציון שמפת הסיביות אינה דחוסה וטבלת הצבעים מורכבת משלוש מסכות צבע של מילים כפולות, אשר מגדירות את האלמנטים אדום, ירוק, וכחול בהתאמה, לכל פיקסל. הערך BI_BITFIELDS תקף עבור התוכניות הפועלות עם מפות סיביות של 16 או 32 סיביות לפיקסל.

המבנה BITMAPINFO מחבר את המבנה BITMAPINFOHEADER ואת טבלת הצבעים, כדי לספק הגדרה שלמה של המימדים והצבעים של מפת הסיביות שאינה תלויית התקן. היישום צריך להשתמש במידע שמכיל האיבר biSize כדי לאתר את טבלת הצבעים במבנה BITMAPINFO, כמו שרואים להלן:

```
pColor = ((LPSTR)pBitmapInfo +
          (WORD)(pBitmapInfo->bmiHeader.biSize));
```

Windows תומכת בפורמטים לדחיסת מפות סיביות שמגדירות את הצבעים שלהן עם 8 או 4 סיביות לפיקסל. הדחיסה מצמצמת את נפח האחסון בדיסק ובזיכרון אשר דרוש עבור מפת הסיביות. בטבלה 7.2 למדת על כך ש-Windows תומכת בשלוש תבניות (פורמטים) לדחיסה.

כאשר ערך האיבר biCompression הוא BI_RLE8, התוכנית היוצרת דוחסת במקור את מפת הסיביות בפורמט קידוד רצף (RLE) עבור מפת סיביות בעלת 8 סיביות לפיקסל. התוכנית היוצרת יכולה להשתמש בפורמט קידוד רצף, כדי לדחוס במצב מקודד (Encoded) או במצב מוחלט (Absolute). שני מצבי העבודה יכולים להתרחש בכל מקום באותה מפת סיביות.

מצב העבודה המקודד (Encoded Mode) מורכב משני בתים. הבית הראשון מגדיר את מספר הפיקסלים העוקבים שצריכה Windows לצייר על ידי השימוש באינדקס הצבע שמוכל בבית השני. בנוסף לכך, התוכנית אשר יצרה את מפת הסיביות יכולה לקבוע את הבית הראשון מזוג בתים אלה לאפס, כדי לציין **חילוף** (Escape) אשר מציין סוף שורה, סוף מפת סיביות, או דלתה (Delta, כלומר, שינוי). הפירוש של החילוף תלוי בערך הבית השני של זוג הבתים, אשר יכול להיות אחד מאלה שמפורטים בטבלה 7.3.

טבלה 7.3: הערכים האפשריים של הבית השני בזוג הבתים.

ערך	פירוש
0	סוף שורה.
1	סוף מפת סיביות.
2	דלתה. שני הבתים שבאים אחרי החילוף מכילים ערכים לא מסומנים (Unsinged Values), אשר מציינים את ההיסט האופקי ואת ההיסט האנכי של הפיקסל הבא בתור, החל מהמקום הנוכחי.

מצד שני, **במצב העבודה המוחלט** (Absolute Mode), הבית הראשון שווה לאפס והבית השני הוא ערך בתחום 03H עד FFH. הבית השני מייצג את מספר הבתים הבאים בתור, שכל אחד מהם מכיל את אינדקס הצבע של פיקסל בודד. כאשר הבית השני שווה ל- 2 או פחות, לחילוף יש פירוש דומה לזה של מצב העבודה המקודד. במצב העבודה המוחלט, התוכנית היוצרת חייבת להסב כל הפעלה במפת סיביות שאינה תלויית התקן על גבול מילה (Word).

אחרי המבנה BITMAPINFOHEADER, מפת הסיביות שאינה תלויית התקן מכילה את טבלת הצבעים. טבלת הצבעים היא קבוצה של מבנים מסוג RGBQUAD אשר מחזיקים את צבע ה-RGB לכל צבע שמשמש את מפת הסיביות. המבנה RGBQUAD מתאר צבע שמורכב מהעוצמות היחסיות של הצבעים אדום, ירוק, וכחול. מספר המבנים מסוג RGBQUAD הוא כמספר הצבעים של מפת הסיביות. Windows API מגדירה את המבנה RGBQUAD כמו שרואים להלן:

```
typedef struct tagRGBQUAD {
        BYTE   rgbBlue;
        BYTE   rgbGreen;
        BYTE   rgbRed;
        BYTE   rgbReserved;
} RGBQUAD;
```

המבנה RGBQUAD מורכב מהאיברים שמפורטים בטבלה 7.4.

טבלה 7.4: איברי המבנה **RGBQUAD**.

איבר	תיאור
rgbBlue	עוצמת הצבע הכחול.
RgbGreen	עוצמת הצבע הירוק.
RgbRed	עוצמת הצבע האדום.
RgbReserved	איבר שמור של Windows; חייב להיות אפס.

החלק האחרון של קובץ מפת הסיביות שאינה תלויית התקן מכיל את נתוני הפיקסלים הממשיים עבור מפת הסיביות. תלמד יותר על יצירה והצגה של מפת סיביות תלויית התקן ועל מפת סיביות שאינה תלויית התקן בסעיפים הבאים.

7.3 יצירת מפות סיביות

קובץ מפת סיביות הוא אחד משלושת הסוגים של קבצי מקור גרפיים של Windows API. באפשרותך ליצור בתוכניות שלך מפות סיביות באמצעות הפונקציה CreateBitmap ולהציג אותן בחלון. פונקציה זו יוצרת מפת סיביות עם הרוחב, הגובה ותבנית הצבע (מישורי צבע וסיביות לכל פיקסל) שמוגדרים על ידך. את הפונקציה CreateBitmap כותבים כמו בהגדרה זו:

```
HBITMAP CreateBitmap(
 int nWidth,           // bitmap width, in pixels
 int nHeight,          // bitmap height, in pixels
 UINT cPlanes,         // number of color planes used by device
 UINT cBitsPerPel,     // number of bits required to identify
                       // a color
 CONST VOID *lpvBits  // pointer to array containing
                       // color data
);
```

כאשר התוכניות קוראות לפונקציה CreateBitmap, פונקציה זו מצפה שיסופקו לה הפרמטרים שמפורטים בטבלה 7.5.

טבלה 7.5: הפרמטרים של הפונקציה **CreateBitmap**.

פרמטר	תיאור
nWidth	רוחב מפת הסיביות בפיקסלים.
NHeight	גובה מפת הסיביות בפיקסלים.
CPlanes	מספר מישורי הצבע שמשתמש בהם ההתקן.
CBitsPerPel	מספר הסיביות שדורש ההתקן כדי לזהות צבע של פיקסל אחד.
LpvBits	מצביע למערך של נתוני צבע שההתקן משתמש בהם, כדי לקבוע את הצבעים במלבן של פיקסלים. אתה חייב ליישר כל קו סריקה שבמלבן לפי גבול מילה (עליך להשלים באפס קווי סריקה שאינם מיושרים לפי מילה). אם פרמטר זה הוא NULL, מפת הסיביות החדשה לא תהיה מוגדרת.

אם הפונקציה CreateBitmap מצליחה, הערך המוחזר הוא ידית למפת הסיביות; אם הפונקציה נכשלת, הערך המוחזר הוא NULL.

לאחר שיוצרים מפת סיביות, באפשרותך לקרוא לפונקציה SelectObject כדי לבחור במפת הסיביות לתוך הקשר התקן. התוכניות יכולות להשתמש בפונקציה CreateBitmap כדי ליצור מפות סיביות בעלות צבע, בגלל סיבות ביצוע, היישומים צריכים להשתמש בפונקציה CreateBitmap כדי ליצור מפות סיביות בצבע אחד (Monochrome Bitmaps), ולהשתמש בפונקציה CreateCompatibleBitmap כדי ליצור מפות סיביות בצבע מלא. כאשר התוכנית בוחרת במפת סיביות בצבע שהוחזרה קודם מ-CreateBitmap לתוך הקשר התקן, Windows חייבת לוודא שמפת הסיביות הזו מתאימה לפורמט של הקשר ההתקן שמקבל אותה. מכיון ש-CreateCompatibleBitmap מקבלת הקשר התקן, היא מחזירה מפת סיביות שיש לה פורמט כמו זה של הקשר ההתקן המוגדר. לכן, קריאות עוקבות ל-SelectObject יותר מהירות מאשר מפות סיביות בצבע שהפונקציה CreateBitmap מחזירה.

אם מפת הסיביות היא בעלת צבע אחד, **צבע הקידמה** (Foreground Color) מיוצג על ידי סיביות "אפס" ו**צבע הרקע** (Background Color) מיוצג על ידי סיביות "אחד", עבור הקשר ההתקן של היעד. אם היישום קובע את הפרמטרים nWidth ו-nHeight לאפס, CreateBitmap מחזירה את הידית של פיקסל בגודל 1 על 1, במפת סיביות בעלת צבע אחד. כאשר אינך צריך יותר את מפת הסיביות, עליך לקרוא לפונקציה DeleteObject כדי למחוק אותה.

כדי להבין יותר טוב את פעולת הפונקציה CreateBitmap, התבונן בתוכנית **Create_Bitmap**, שבתקליטור המצורף לספר זה. התוכנית יוצרת מפת סיביות בעלת צבע אחד כאשר המשתמש בוחר באפשרות !Test מהתפריט. אחר כך היא מעבירה את מפת הסיביות לתוך הקשר התקן שבזיכרון ומציירת מלבן ואליפסה במפת הסיביות. לבסוף, התוכנית מציגה את מפת הסיביות שמתקבלת. התוכנית מבצעת את העיבוד העיקרי שלה בפונקציה WndProc.

7.4 הצגת מפות סיביות

התוכנית חייבת "לדחוף" את מפת הסיביות לתוך הקשר התקן התצוגה, כדי שניתן יהיה להציג אותה. בתוכנית **Create_Bitmap** יצרנו תחילה את מפת הסיביות, ואחר כך הוספנו אותה להקשר ההתקן כדי להציגה. כרגיל, התוכניות משתמשות בפונקציה BitBlt להצגת מפת סיביות. פונקציה זו מעבירה בלוק סיביות של נתוני הצבע המתייחסים למלבן של פיקסלים, מהקשר התקן המקור שאתה מגדיר אל הקשר התקן היעד. עליך להעביר תחילה את מפת הסיביות אל הקשר התקן בזיכרון (אשר אתה יוצר עם CreateCompatibleDC). אחר כך, עליך לקרוא לפונקציה BitBlt כדי להעתיק את מפת הסיביות להקשר ההתקן הממשי. מפני שנוהל ההעתקה הממשי של BitBlt משתמש **בפעולות רשת** (Raster Operations), אתה צריך לבדוק את ה-RASTERCAPS של ההתקן לפני שמפעילים את BitBlt כנגד ההתקן.

את הפונקציה BitBlt כותבים כמו בהגדרה זו:

```
BOOL BitBlt(
  HDC hdcDest,  // handle to destination device context
  int nXDest,   // x-coordinate of destination
                // rectangle's upper-left corner
  int nYDest,   // y-coordinate of destination
                // rectangle's upper-left corner
  int nWidth,   // width of destination rectangle
  int nHeight,  // height of destination rectangle
  HDC hdcSrc,   // handle to source device context
  int nXSrc,    // x-coordinate of source rectangle's
                // upper-left corner
  int nYsrc,    // y-coordinate of source rectangle's
                // upper-left corner
  DWORD dwRop   // raster operation code
);
```

הפונקציה BitBlt מקבלת את הפרמטרים שמפורטים בטבלה 7.6.

טבלה 7.6: הפרמטרים של הפונקציה **BitBlt**.

פרמטר	תיאור
hdcDest	מזהה את הקשר התקן היעד.
nXDest	מגדיר את הקואורדינטה הלוגית X (Logical X-Coordinate) של הפינה השמאלית העליונה של מלבן היעד.
nYDest	מגדיר את הקואורדינטה הלוגית Y (Logical Y-Coordinate) של הפינה השמאלית העליונה של מלבן היעד.
nWidth	מגדיר את הרוחב הלוגי של מלבן המקור ושל מלבן היעד.
nHeight	מגדיר את הגובה הלוגי של מלבן המקור ושל מלבן היעד.
hdcSrc	מזהה את הקשר התקן המקור.
nXSrc	מגדיר את הקואורדינטה הלוגית X (Logical X-Coordinate) של הפינה השמאלית העליונה של מלבן המקור.
nYSrc	מגדיר את הקואורדינטה הלוגית Y (Logical Y-Coordinate) של הפינה השמאלית העליונה של מלבן המקור.
dwRop	מזהה את קוד פעולות הרשת (Raster Operation). קודים אלה מגדירים איך הפונקציה צריכה לשלב את נתוני הצבע של מלבן המקור עם נתוני הצבע של מלבן היעד, כדי ליצור את הצבע הסופי הרצוי. טבלה 7.7 מפרטת חלק מקודי הרשת הנפוצים.

למדת שהתוכניות מבצעות פעולות רשת על מפות הסיביות כאשר ממקמים אותן בתוך הקשר התקן. טבלה 7.7 מציגה חלק מקודי הרשת הנפוצים, אשר משמשים את התוכניות כאשר ממקמים מפות הסיביות בהקשר התקן.

טבלה 7.7: פעולות רשת (Raster Operations) נפוצות אשר יבוצעו על ידי התוכניות בעבודה עם מפות סיביות.

ערך	תיאור
BLACKNESS	משתמש בצבע הקשור עם אינדקס אפס שבלוח הצבעים הפיזי , כדי למלא את מלבן היעד (צבע זה הוא שחור עבור ברירת המחדל של לוח הצבעים).
DSTINVERT	הופך את מלבן היעד.
MERGECOPY	משתמש באופרטור הבוליאני AND כדי למזג את הצבעים של מלבן המקור עם התבנית המוגדרת (Specified Pattern).
MERGEPAINT	משתמש באופרטור הבוליאני OR כדי למזג את הצבעים של מלבן המקור שעבר היפוך (Inverted Source Rectangle) עם הצבעים של מלבן היעד.
NOTSRCCOPY	מעתיק אל היעד את מלבן המקור שעבר היפוך.
NOTSRCERASE	משתמש באופרטור הבוליאני OR כדי לשלב את הצבעים של מלבני המקור והיעד, ואחר כך הופך את הצבע שמתקבל.
PATCOPY	מעתיק את התבנית המוגדרת למפת סיביות היעד.
PATINVERT	משתמש באופרטור הבוליאני XOR כדי לחבר את הצבעים של התבנית המוגדרת עם הצבעים של מלבן היעד.
PATPAINT	משתמש באופרטור הבוליאני OR, כדי לשלב את צבעי התבנית עם הצבעים של מלבן המקור שעבר היפוך. באפשרותך להשתמש באופרטור הבוליאני OR כדי לשלב את תוצאת פעולה זו עם צבעי מלבן היעד.
SRCAND	משתמש באופרטור הבוליאני AND כדי לשלב את הצבעים של מלבני המקור והיעד.
SRCCOPY	מעתיק את מלבן המקור ישירות למלבן היעד.
SRCERASE	משתמש באופרטור הבוליאני AND כדי לשלב את צבעי מלבן היעד שעברו היפוך עם הצבעים של מלבן המקור.
SRCINVERT	משתמש באופרטור הבוליאני XOR כדי לשלב את הצבעים של מלבני המקור והיעד.
SRCPAINT	משתמש באופרטור הבוליאני OR כדי לחבר את הצבעים של מלבני המקור והיעד.
WHITENESS	משתמש בצבע הקשור עם אינדקס 1 שבלוח הצבעים הפיזי כדי למלא את מלבן היעד (צבע זה הוא לבן עבור ברירת המחדל של לוח הצבעים).

אם טרנספורמציות סיבוב או גזירה (Shear), טרנספורמציה אשר משנה את האורך והכיוון הנראים של קווים אנכיים או אופקיים באובייקט) נמצאת בפעולה בהקשר ההתקן של המקור, BitBlt מחזירה ציון שגיאה. אם יש טרנספורמציות אחרות בהקשר התקן המקור (והטרנספורמציה המתאימה אינה מופעלת בהקשר ההתקן של היעד), הפונקציה BitBlt מותחת, דוחסת, או מסובבת את המלבן שבהקשר ההתקן של היעד כפי שדרוש.

אם תבניות הצבע של הקשר התקן המקור ושל הקשר התקן היעד אינן תואמות, הפונקציה BitBlt מתאימה את תבנית צבע המקור לזו של תבנית היעד. כאשר BitBlt כותבת קובץ-על משופר (Enhanced Metafile), עלולה להיות שגיאה אם הקשר ההתקן של המקור מזהה הקשר התקן של קובץ-על משופר. BitBlt מחזירה שגיאה אם הקשרי ההתקן של המקור והיעד מייצגים התקנים שונים (כלומר צריך תמיד ליצור את הקשר המקור עם הפונקציה CreateCompatibleDC).

כדי להבין יותר טוב כיצד הפונקציה BitBlt פועלת כחלק מתוכנית, התבונן בתוכנית **Happy_Faces** שבתקליטור המצורף לספר זה (בתיקיה Chap07). התוכנית טוענת מפת סיביות לזיכרון, ואחר כך מעבירה אותה אל שטח הלקוח של החלון. החלק הפעיל של התוכנית נמצא בפונקציה WndProc.

7.5 יצירת מפות סיביות תלויות התקן (DIB)

Windows תומכת במפות סיביות תלויות התקן ובמפות סיביות שאינן תלויות התקן. בסעיף קודם יצרנו מפת סיביות תלויית התקן והצגנו אותה על המסך. הצגת מפות סיביות שאינן תלויות התקן נעשית בדרך דומה. חובה להפוך כל מפת סיביות שאינה תלויית התקן למפת סיביות תלויית התקן, כדי שנוכל להציג אותה בהקשר התקן. משתמשים בפונקציה CreateDIBitmap כדי ליצור מפת סיביות תלויית התקן מתוך מפת סיביות שאינה תלויית התקן, ואפשר גם לקבוע את הסיביות של מפת הסיביות. את הפונקציה CreateDIBitmap כותבים כמו כמו בהגדרה שלהלן:

```
HBITMP CreateDIBitmap(
  HDC hdc,              // handle to device context
  CONST BITMAPINFOHEADER *lpbmih,  // pointer to bitmap
                        // size and format data
  DWORD fdwInit,        // initialization flag
  CONST VOID *lpbInit,  // pointer to initialization data
  CONST BITMAPINFO *lpbmi, // pointer to bitmap
                        // color-format data
  UINT fuUsage          // color-data usage
);
```

הפונקציה CreateDIBitmap מקבלת את הפרמטרים שמפורטים בטבלה 7.8.

טבלה 7.8 : הפרמטרים של הפונקציה CreateDIBitmap.

פרמטר	תיאור
hdc	מזהה את הקשר ההתקן.
lpbmih	מצביע למבנה מסוג BITMAPINFOHEADER. אם fdwInit שווה לערך CBM_INIT, הפונקציה תשתמש במבנה BITMAPINFOHEADER כדי להשיג את הרוחב והגובה הרצויים של מפת הסיביות ומידע אחר. שים לב שערך חיובי עבור הגובה מציין מפת סיביות שאינה תלויית התקן מלמטה-למעלה, וערך שלילי עבור הגובה מציין מפת סיביות שאינה תלויית התקן מלמעלה-למטה. תסריט זה תואם את הפונקציה CreateDIBitmap.
fdwInit	קבוצה של דגלי סיביות אשר מגדירים כיצד מערכת ההפעלה מאתחלת את סיביות מפת הסיביות. התוכנית יכולה להגדיר קבוע אחד בלבד לפרמטר זה. ערך הפרמטר חייב להיות CBM_INIT או אפס. אם דגל זה נקבע, מערכת ההפעלה משתמשת בנתונים שהפרמטרים lpbInit ו- lpbmi מצביעים עליהם, כדי לאתחל את סיביות מפת הסיביות. אם דגל זה אינו נקבע, הפונקציה אינה משתמשת בנתונים שמוצבעים על ידי פרמטרים אלה. אם fdwInit שווה לאפס, מערכת ההפעלה אינה מאתחלת את סיביות מפת הסיביות.
lpbInit	מצביע למערך של בתים, שמכיל את נתוני האתחול של מפת הסיביות. הפורמט של הנתונים תלוי באיבר biBitCount של המבנה BITMAPINFO אשר הפרמטר lpbmi מצביע עליו.
lpbmi	מצביע למבנה מסוג BITMAPINFO אשר מתאר את המימדים ותבנית הצבע של המערך, אשר הפרמטר lpbInit מצביע עליו.
fuUsage	מגדיר אם התוכנית אשר ייצרה את מפת הסיביות גם מאתחלת את האיבר bmiColors של המבנה BITMAPINFO ; ואם כן, האם bmiColors מכיל ערכים מפורשים של אדום, ירוק וכחול (RGB), או אינדקסים של לוח צבעים. הפרמטר fuUsage חייב להכיל ערך אחד או יותר מהערכים שמפורטים בטבלה 7.9.

הפרמטר fuUsage מקבל ערך קבוע אחד מתוך שני ערכים אפשריים. טבלה 7.9 מפרטת את ערכי הקבועים האפשריים של הפרמטר fuUsage.

טבלה 7.9 הערכים האפשריים של הפרמטר **fuUsage**.

ערך	פירוש
DIB_PAL_COLORS	קובץ מפת הסיביות מספק טבלת צבעים שמורכבת ממערך של אינדקסים בני 16 סיביות בלוח הצבעים הלוגי של הקשר ההתקן, אשר התוכנית מציבה לתוכו את מפת הסיביות.
DIB_RGB_COLORS	קובץ מפת הסיביות מספק טבלת צבעים ומכיל ערכי RGB.

אם הפונקציה מצליחה, הערך המוחזר הוא ידית למפת הסיביות; אם הפונקציה נכשלת, הערך המוחזר הוא NULL.

כדי להבין יותר טוב את פעולת הפונקציה CreateDIBitmap, התבוננו בתוכנית **Create_DIB_Bitmap** שבתקליטור המצורף לספר זה (בתיקיה Chap07). התוכנית טוענת מפת סיביות שאינה תלויית התקן, מציירת את מפת הסיביות בהקשר התקן בזיכרון, ואחר כך מתרגמת את הקשר ההתקן שבזיכרון להקשר ההתקן של המסך (חלון התוכנית). כרגיל, קוד התוכנית שבפונקציה WndProc של התוכנית **Create_DIB_Bitmap** מבצע את העיבוד המעשי.

7.6 מילוי מלבן בתבנית

תוכניות משתמשות במפות סיביות, כדי להוסיף גרפיקה לתצוגת החלון. בנוסף לכך, תוכניות יכולות להשתמש בעטים ובהקשרי התקן בזיכרון, כדי לצייר בחלון צורות פשוטות או מורכבות. לדוגמה, התוכניות יכולות להשתמש בפונקציה PatBlt כדי לצייר מלבנים בהקשר התקן. הפונקציה PatBlt מציירת את המלבן הנתון לתוך הקשר ההתקן באמצעות המברשת הנוכחית, זו שנבחרה לאחרונה. הפונקציה PatBlt משתמשת בפעולת הרשת (Raster Operation) הנתונה כדי לשלב את צבע המברשת ואת צבע המשטח. את הפונקציה PatBlt כותבים כמו בהגדרה שלהלן:

```
BOOL PatBlt(
  HDC hdc,           // handle to device context
  int nXLeft,        // x-coord. of upper-left corner of
                     // rect. to be filled
  int nYLeft,        // y-coord. of upper-left corner of
                     // rect. to be filled
  int nWidth,        // width of rectangle to be filled
  int nHeight,       // height of rectangle to be filled
  DWORD dwRop        // raster operation code
);
```

הפונקציה PatBlt מקבלת את הפרמטרים שמפורטים בטבלה 7.10 .

טבלה 7.10: הפרמטרים של הפונקציה **PatBlt**.

פרמטר	תיאור
hdc	מזהה את הקשר ההתקן.
nXLeft	מגדיר ביחידות לוגיות את קואורדינטת X של הפינה השמאלית העליונה של המלבן אשר הפונקציה עומדת למלא.
nYLeft	מגדיר ביחידות לוגיות את קואורדינטת Y של הפינה השמאלית העליונה של המלבן אשר הפונקציה עומדת למלא.
nWidth	מגדיר ביחידות לוגיות את רוחב המלבן.
nHeight	מגדיר ביחידות לוגיות את גובה המלבן.
dwRop	מגדיר את קוד פעולת הרשת (Raster Operation Code). קוד זה יכול להיות אחד הערכים שמפורטים בטבלה 7.11.

כמו שראית בטבלה זו, הפרמטר dwRop מקבל מספר שונה של קודי פעולות רשת. טבלה 7.11 מפרטת את קודי פעולות הרשת השונים, אשר התוכנית תשתמש בהם.

טבלה 7.11: הערכים האפשריים של קודי רשת.

ערך	תיאור
PATCOPY	מעתיק אל מפת סיביות היעד את התבנית המוגדרת.
PATINVERT	משתמש באופרטור הבוליאני OR כדי לשלב את צבעי התבנית המוגדרת ואת צבעי מלבן היעד.
DSTINVERT	הופך את מלבן היעד.
BLACKNESS	משתמש בצבע הקשור עם אינדקס אפס שבלוח הצבעים הפיזי, כדי למלא את מלבן היעד (צבע זה הוא שחור עבור ברירת המחדל של לוח הצבעים).
WHITENESS	משתמש בצבע הקשור עם האינדקס 1 שבלוח הצבעים הפיזי, כדי למלא את מלבן היעד (צבע זה הוא לבן עבור ברירת המחדל של לוח הצבעים).

ערכי הפרמטר dwRop של הפונקציה PatBlt הם תת-קבוצה מוגבלת מהקוד המלא של 256 הפעולות המתייחסות לרשת תלת-מימדית (Ternary Raster); במיוחד, אינך יכול להשתמש בקוד פעולה אשר מתייחס למלבן מקור. לעיון ברשימה המלאה של קודי הפעולות עבור רשת תלת-מימדית פנה אל העזרה המקוונת של המהדר שלך. דע, שלא כל ההתקנים תומכים בפונקציה PatBlt. צריך להשתמש בפונקציה GetDeviceCaps כדי לבדוק את תאימות RC_BITBLT של ההתקן, לפני שאתה קורא לפונקציה PatBlt עבור ההתקן.

כדי להבין יותר טוב את פעולת הפונקציה PatBlt, התבונן בתוכנית **Paint_Rect** שבתקליטור המצורף לספר זה (בתיקיה Chap07). התוכנית מציירת בחלון התוכנית תיבה אפורה עם גבול תלת-מימדי. התוכנית **Paint_Rect** משתמשת בשלוש מברשות סטנדרטיות כדי לצייר את המלבן שמרכיב את הגבול התלת-ממדי של התיבה. הפונקציה WndProc של התוכנית **Paint_Rect** מבצעת את העיבוד המעשי.

7.7 שימוש ב-SetDIBits

ככל שהתוכניות הופכות ליותר מורכבות, ייתכן לעיתים צורך לשנות את **ערכת הצבעים של מפת הסיביות** (Bitmap's Color Scheme). למדת שמפות סיביות שאינן תלויות התקן מחזיקות את מידע הצבע בכותר מפת הסיביות (Bitmap's Header). באפשרותך להשתמש בצבעים שהתוכנית היוצרת אחסנה קודם לכן בכותר מפת הסיביות שאינה תלויית התקן, כדי לשנות את הצבעים של מפת סיביות נתונה. הפונקציה SetDIBits משתמשת בנתוני הצבע שהיא מוצאת במפת הסיביות המוגדרת שאינה תלויית התקן, כדי לקבוע את הפיקסלים במפת הסיביות. את הפונקציה SetDIBits כותבים בתוכניות כמו בהגדרה שלהלן:

```
int SetDIBits(
  HDC hdc,                    // handle to device context
  HBITMAP hbmp,               // handle of bitmap
  UINT uStartScan,            // starting scan line
  UINT cScanLines,            // number of scan lines
  CONST VOID *lpvBits,        // array of bitmap bits
  CONST BITMAPINFO *lpbmi,    // address of structure with
                              // bitmap data
  UINT fuColorUse             // type of color indices to use
);
```

הפונקציה SetDIBits מקבלת את הפרמטרים שמפורטים בטבלה 7.12.

טבלה *7.12*: הפרמטרים של הפונקציה **SetDIBits**.

פרמטר	תיאור
hdc	מזהה את הקשר התקן.
hbmp	מזהה את מפת הסיביות אשר הפונקציה SetDIBits עומדת לשנות על ידי שימוש בנתוני הצבע ממפת הסיביות שאינה תלויית התקן.
uStartScan	מגדיר את קו הסריקה ההתחלתי עבור נתוני הצבע שאינו תלוי התקן במערך שמוצבע על ידי הפרמטר lpvBits.
cScanLines	מגדיר את מספר קווי הסריקה שמצאה הפונקציה במערך שמכיל נתוני צבע שאינם תלויים בהתקן.

פרמטר	תיאור
lpvBits	מצביע לנתוני הצבע של מפת הסיביות שאינה תלויית התקן, אשר הפונקציה מאחסנת כמערך בתים. התבנית של ערכי מפת הסיביות תלויה באיבר biBitCount של המבנה BITMAPINFO שמוצבע על ידי הפרמטר lpbmi.
lpbmi	מצביע למבנה הנתונים BITMAPINFO שמכיל מידע על מפת הסיביות שאינה תלויית התקן.
fuColorUse	מגדיר אם הגדרת מפת הסיביות מכילה את האיבר bmiColors של המבנה BITMAPINFO ; ואם כן, האם bmiColors מכיל ערכים מפורשים של אדום, ירוק, כחול (RGB), או אינדקסי צבע של לוח צבעים. הפרמטר fuColorUse חייב להיות אחד הערכים שמפורטים בטבלה 7.13.

בתוך טבלה 7.12 אפשר ללמוד שהפרמטר fuColorUse מקבל מספר ערכים קבועים מובנים. טבלה 7.13 מפרטת את הערכים שמקבל הפרמטר fuColorUse.

טבלה 7.13: הערכים האפשריים של הפרמטר **fuColorUse**.

ערך	פירוש
DIB_PAL_COLORS	טבלת הצבעים מורכבת ממערך של אינדקסים בני 16 סיביות בלוח הצבעים הלוגי של הקשר ההתקן שמזוהה על ידי הפרמטר hdc.
DIB_RGB_COLORS	מפת הסיביות מספקת טבלת צבעים שמכילה במפורש ערכי RGB.

אם הפונקציה מצליחה, הערך המוחזר הוא מספר קווי הסריקה אשר הפונקציה SetDIBits העתיקה בהצלחה; אם הפונקציה נכשלת, הערך המוחזר הוא אפס. Windows משיגה מהירות אופטימלית בציור מפת סיביות כאשר הסיביות של מפת הסיביות הם אינדקסים בלוח הצבעים של המערכת.

יישום יכול לקרוא לפונקציה GetSystemPaletteEntries כדי לקבל את צבעי לוח הצבעים של המערכת ושל האינדקסים. לאחר שהיישום מקבל את הצבעים והאינדקסים, הוא יכול ליצור את מפת הסיביות שאינה תלויית התקן. הפונקציה משתמשת בהקשר ההתקן שמזוהה על ידי הפרמטר hdc רק אם אתה מציב את הקבוע DIB_PAL_COLORS עבור הפרמטר fuColorUse; אחרת, הפונקציה מתעלמת מהפרמטר hdc.

לתוכנית שלך או לתוכנית אחרת אסור לבחור את מפת הסיביות שמזוהה על ידי הפרמטר hbmp לתוך הקשר ההתקן, בשעה שהיישום קורא לפונקציה GetSystemPaletteEntries. נקודת המוצא של מפות סיביות שאינן תלויות התקן מלמטה-למעלה היא הפינה השמאלית התחתונה של מפת הסיביות; נקודת המוצא של מפות סיביות שאינן תלויות התקן מלמעלה-למטה היא הפינה השמאלית העליונה של מפת הסיביות.

כדי להבין יותר טוב את פעולת הפונקציה SetDIBits, התבונן בתוכנית **Change_Colors** שנמצאת התקליטור המצורף (בתיקיה Chap07). תוכנית זו מציירת מפת סיביות שאינה תלויית התקן בשטח הלקוח של החלון. בתחילה, התוכנית צובעת את מפת הסיביות בצבעי שחור ולבן. כאשר המשתמש בוחר באפשרות Test! מהתפריט, התוכנית משנה את נתוני מפת הסיביות, כך שכל שני פיקסלים שחורים עוקבים מחזירים את הצירוף של פיקסל כחול-שחור. הפונקציה WndProc מכילה את הקוד המעשי של התוכנית.

7.8 פלט של מפת סיביות להתקן נתון באמצעות SetDIBitsToDevice

למדת שבאפשרות התוכניות להשתמש בפונקציה SetDIBits כדי לקבוע את סיביות הצבע של מפת סיביות, בהתאם לערכים שמכיל הכותר של מפת הסיביות שאינה תלויית התקן. פעמים רבות, התוכניות חייבות לצמצם את מספר הצבעים של מפת סיביות שאינה תלויית התקן בשעה שהן מפיקות פלט אל התקן מסוים. לדוגמה, Windows חייבת למפות מפת סיביות של 256 צבעים למפת סיביות של 20 צבעים כאשר צריך להוציא את מפת הסיביות כפלט אל צג VGA. הפונקציה SetDIBitsToDevice מציירת מפת סיביות שאינה תלויית התקן בהתקן שקשור עם הקשר ההתקן הנתון. בנוסף לכך, הפונקציה משתמשת בנתוני הצבע של מפת הסיביות שאינה תלויית התקן המקורית, כדי לקבוע את הפיקסלים במפת הסיביות המצוירת.

את הפונקציה SetDIBitsToDevice כותבים בתוכניות כמו בהגדרה שלהלן:

```
int SetDIBitsToDevice(
  HDC hdc,                // handle of device context
  int XDest,              // x-coordinate of upper-left
                          // corner of dest. rect.
  int YDest,              // y-coordinate of upper-left
                          // corner of dest. rect.
  DWORD dwWidth,          // source rectangle width
  DWORD dwHeight,         // source rectangle height
  int XSrc,               // x-coordinate of lower-left
                          // corner of source rect.
  int YSrc,               // y-coordinate of lower-left
                          // corner of source rect.
  UINT uStartScan,        // first scan line in array
  UINT cScanLines,        // number of scan lines
  CONST VOID *lpvBits,    // address of array with DIB bits
  CONST BITMAPINFO *lpbmi,// address of structure with
                          // bitmap info.
  UINT fuColorUse         // RGB or palette indices
);
```

הפונקציה SetDIBitsToDevice מקבלת את הפרמטרים שמפורטים בטבלה 7.14.

טבלה 7.14: הפרמטרים של הפונקציה SetDIBitsToDevice.

פרמטר	תיאור
hdc	מזהה את הקשר ההתקן.
XDest	מגדיר ביחידות לוגיות את קואורדינטת X (X-Coordinate) של הפינה השמאלית העליונה של מלבן היעד.
YDest	מגדיר ביחידות לוגיות את קואורדינטת Y (Y-Coordinate) של הפינה השמאלית העליונה של מלבן היעד.
dwWidth	מגדיר ביחידות לוגיות את הרוחב של מפת הסיביות שאינה תלויית התקן.
dwHeight	מגדיר ביחידות לוגיות את הגובה של מפת הסיביות שאינה תלויית התקן.
XSrc	מגדיר ביחידות לוגיות את קואורדינטת X של הפינה השמאלית התחתונה של מפת הסיביות שאינה תלויית התקן.
YSrc	מגדיר ביחידות לוגיות את קואורדינטת Y של הפינה השמאלית התחתונה של מפת הסיביות שאינה תלויית התקן.
uStartScan	מגדיר את קו הסריקה ההתחלתי במפת הסיביות שאינה תלויית התקן.
cScanLines	מגדיר את מספר קווי הסריקה במפת הסיביות שאינה תלויית התקן, במערך שמוצבע על ידי הפרמטר lpvBits.
lpvBits	מצביע לנתוני הצבע של מפת הסיביות שאינה תלויית התקן, אשר התוכנית היוצרת של מפת הסיביות אחסנה אותם קודם לכן כמערך בתים.
lpbmi	מצביע למבנה הנתונים BITMAPINFO שמכיל מידע על מפת הסיביות שאינה תלויית התקן.
fuColorUse	מגדיר אם האיבר bmiColors של המבנה BITMAPINFO מכיל ערכים מפורשים של אדום, ירוק, כחול (RGB), או אינדקסי צבע של לוח צבעים. הפרמטר fuColorUse חייב להיות אחד הערכים שמפורטים בטבלה 7.13.

ניתן לראות, שהפרמטר fuColorUse מקבל אחד משני ערכים מובנים. טבלה 7.13 מפרטת את הערכים שמקבל הפרמטר fuColorUse עם הפונקציה SetDIBitsToDevice.

הפונקציה משיגה מהירות אופטימלית בציור מפת סיביות כאשר הסיביות של מפת הסיביות הם אינדקסים בלוח הצבעים של המערכת. היישום יכול לקרוא לפונקציה GetSystemPaletteEntries כדי לקבל את צבעי לוח הצבעים של המערכת ואת האינדקסים. לאחר שהיישום מקבל את הצבעים והאינדקסים, הוא יכול ליצור את מפת הסיביות שאינה תלויית התקן.

נקודת המוצא של מפות סיביות שאינן תלויות התקן מלמטה-למעלה היא הפינה השמאלית התחתונה של מפת הסיביות; נקודת המוצא של מפות סיביות שאינן תלויות התקן מלמעלה-למטה היא הפינה השמאלית העליונה של מפת הסיביות. על התוכנית לצמצם את כמות הזיכרון שדרושה לקביעת סיביות במשטח ההתקן מתוך מפת סיביות גדולה שאינה תלויית התקן. לשם כך, היא יכולה לחזור ולקרוא ל- SetDIBitsToDevice כדי לחלק את הפלט, פעולה שמציבה כל פעם חלקים שונים ממפת הסיביות אל תוך המערך lpvBits. ערכי הפרמטרים uStartScan ו- cScanLines מזהים את החלק של מפת הסיביות שמכיל המערך lpvBits. הפונקציה SetDIBitsToDevice מחזירה שגיאה אם תהליך אשר מופעל ברקע (Background) קורא לה בזמן שמסך MS-DOS מלא מופעל בקידמה (Foreground).

כדי להבין יותר טוב את פעולת הפונקציה SetDIBitsToDevice, התבונן בתוכנית **Draw_2_Boxes** שבתקליטור המצורף לספר זה (בתיקיה Chap07). התוכנית משתמשת במפת סיביות שאינה תלויית התקן ובלוח צבעים מותאם אישית. התוכנית מציירת את מפת הסיביות שאינה תלויית התקן בגודלה הנורמלי, ומשתמשת לצורך זה בפונקציה SetDIBitsToDevice ואחר כך משתמשת בפונקציה StretchDIBits כדי להציג את מפת הסיביות ב- 200 אחוז מגודלה המקורי. העיבוד המעשי של התוכנית **Draw_2_Boxes** נעשה בפונקציות הטיפול בהודעות WM_CREATE ו- WM_PAINT שבפונקציה WndProc.

7.9 קבצי-על (Metafiles)

בסעיפים קודמים למדת על מפות סיביות. גם למדת ש- Windows תומכת בשלושה סוגי גרפיקה בסיסיים. הסוג השני ידוע כ- **קובץ-על** (Metafile). קבצי-על מכילים למעשה קריאות מקודדות לפונקציות של ממשק ההתקן הגרפי (Coded Graphic Device Interface Function Calls). כאשר תוכנית **מפעילה** (Plays) קובץ-על, התוצאה זהה, כאילו התוכנית הייתה משתמשת ישירות בפונקציות ממשק ההתקן הגרפי. למעשה, תוכל לראות קובץ-על כמאקרו גרפי יעיל. באפשרותך לאחסן קבצי-על בזיכרון (או כקובץ בדיסק), באפשרותך לטעון מחדש את קובץ-העל, וגם תוכל להריץ את קובץ-העל במספר כלשהו של יישומים שונים. בנוסף לכך, קבצי-על הם אינם תלויי התקן יותר מאשר מפות סיביות, מכיון שהמחשב (וההתקן) מפרשים את פונקציות ממשק ההתקן הגרפי בהתבסס על הקשר התקן הפלט.

Windows 9x ו- Windows NT תומכות בקבצי-העל של Windows 9x. עם זאת, שתי מערכות ההפעלה של Win32 תומכות גם כן בסוג החדש של קובץ-על משופר (Enhanced Metafile), וכך גם התוכניות שלך, צריכות להשתמש בקבצי-על משופרים. ההבדל העיקרי בין קובץ-על של Windows 9x לבין קובץ-על משופר הוא בכך שקבצי-על משופרים באמת אינם תלויים בהתקן וגם תומכים בפונקציות החדשות של ממשק ההתקן הגרפי של Win32 API (Win32 GDI API).

חשוב להבין שממשק Win32 API תומך באופן מלא בקבצי-העל של Win16. למעשה, Win32 API מכיל שתי קבוצות של פונקציות המיועדות לקבצי-על - קבוצה אחת עבור Win16 API וקבוצה שנייה עבור Win32 API. לדוגמה, התוכניות יכולה לקרוא לפונקציה PlayMetaFile כדי להריץ קובץ-על בן 16 סיביות. בדרך דומה, התוכניות יכולה לקרוא לפונקציה PlayEnhMetaFile כדי להריץ קובץ-על בן 32 סיביות. ההסברים שנציג בספר זה מתמקדים בקבצי-על משופרים, ולא בקבצי-על של Win16.

7.10 יצירה והצגה של קבצי-על (Metafiles)

למדת שקובץ-על הוא סדרת הוראות של ממשק התקן גרפי שהתוכנית היוצרת אחסנה קודם לכן במבנה כלשהו. השתמש בפונקציות של ממשק ההתקן הגרפי בהקשר התקן של קובץ-על, כדי ליצור קבצי-על. השתמש **בהקשר התקן מיוחס** (Reference Device Context) כבסיס עבור קובץ-העל שבו תחזיק מימדי תמונה עבור התקני פלט שונים. ההתקן המיוחס מתאים להתקן שבו מוצגת התמונה לראשונה. משתמשים בפונקציה CreateEnhMetaFile כדי ליצור קבצי-על. פונקציה זו יוצרת הקשר התקן עבור מבנה, או פורמט, משופר של קובץ-על. באפשרותך להשתמש בהקשר ההתקן הנוצר כדי לאחסן תמונה שאינה תלויית התקן. את הפונקציה CreateEnhMetaFile כותבים בתוכנית, כמו שניתן לראות בהגדרה שלהלן:

```
HDC CreateEnhMetaFile(
  HDC hdcRef,            // handle to reference device context
  LPCTSTR lpFilename,    // pointer to a filename string
  CONST RECT *lpRect,    // pointer to a bounding rectangle
  LPCTSTR lpDescription  // pointer to an optional
                         // description string
);
```

הפונקציה CreateEnhMetafile מקבלת את הפרמטרים שמפורטים בטבלה 7.15.

טבלה 7.15: הפרמטרים של הפונקציה CreateEnhMetafile.

פרמטר	תיאור
hdcRef	מזהה הקשר התקן מיוחס עבור קובץ-על המשופר.
lpFilename	מצביע לשם הקובץ של קובץ-על משופר שהפונקציה צריכה ליצור. אם פרמטר זה הוא NULL, קובץ-העל המשופר יהיה מבוסס זיכרון והתוכן שלו יאבד כאשר משתמשים בפונקציה DeleteEnhMetafile, כדי למחוק אותו.
lpRect	מצביע למבנה מסוג RECT אשר מגדיר את מימדי התמונה (ביחידות של 0.01 מילימטר) שהתוכנית תאחסן בסופו של דבר בקובץ-העל המשופר.

פרמטר	תיאור
lpDescription	מצביע למחרוזת אשר מגדירה את שם היישום, אשר יצר את התמונה ובכלל זה גם כותרת התמונה. המחרוזת שמצביע עליה הפרמטר lpDescription חייבת להכיל תו NULL בין שם היישום לבין שם התמונה וחייבת להסתיים בשני תווי NULL. לדוגמה המחרוזת "XYZ Graphics Editor\0Bald Eagle\0\0", שבה 0\ מסמן את התו NULL. אם lpDescription הוא NULL, אין כניסה מתאימה בכותר של קובץ-העל המשופר.

Windows משתמשת בהתקן המיוחס שמזוהה על ידי הפרמטר hdcRef כדי לרשום את הרזולוציה ואת היחידות של ההתקן שבו מוצגת התמונה במקור. אם הפרמטר hdcRef הוא NULL, Windows משתמשת בהתקן התצוגה הנוכחי לצורך ייחוס.

האיברים Left ו-Top של המבנה RECT שמוצבעים על ידי הפרמטר lpRect חייבים להיות קטנים מהאיברים Right ו-Bottom, בהתאמה. הנקודות לאורך צלעות המלבן נכללות בתמונה. אם הפרמטר lpRect הוא NULL, ממשק ההתקן הגרפי מחשב את המימדים של המלבן הקטן ביותר שמקיף את התמונה שצויירה על ידי היישום. התוכניות צריכה לספק את הפרמטר lpRect ככל שהדבר אפשרי.

יישומים משתמשים בהקשר ההתקן שנוצר על ידי הפונקציה CreateEnhMetaFile כדי לאחסן תמונה גרפית בקובץ-על משופר. התוכניות יכולה למסור את הידית אשר מזהה את הקשר ההתקן הזה לכל פונקציה של ממשק התקן גרפי.

לאחר שיישום מאחסן תמונה בקובץ-על משופר, הוא יכול לקרוא לפונקציה PlayEnhMetaFile כדי להציג את התמונה בהתקן פלט כלשהו. כדי להציג את התמונה, Windows משתמשת במלבן שמוצבע על ידי הפרמטר lpRect ובנתוני הרזולוציה מההתקן המיוחס כדי למקם ולדרג את התמונה. הקשר ההתקן שמחזירה הפונקציה PlayEnhMetaFile מכיל את אותם ערכי ברירת המחדל שקשורים עם כל הקשר התקן חדש כלשהו. יישומים חייבים להשתמש בפונקציה GetWinMetaFileBits כדי להמיר קובץ-על משופר לפורמט Windows הישן של קובץ-על. שם הקובץ של קובץ-על משופר צריך להשתמש בסיומת EMF.

כדי להבין יותר טוב את פעולת הפונקציה CreateEnhMetaFile, התבונן בתוכנית **Crosshatch_Box** שנמצאת בתקליטור המצורף לספר זה (בתיקיה Chap07). התוכנית יוצרת קובץ-על משופר של מלבן שממולא במברשת מסוג Cross-Hatched (מילוי על ידי תבנית רשת) בעת האתחול. התוכנית מריצה אחר כך את קובץ-העל מתוך הפונקציה לטיפול בהודעות WM_PAINT. התוכנית מדרגת את קובץ-העל בהתבסס על שטח הלקוח של החלון, ומשמעות הדבר שגודל המלבן משתנה ככל שמשנים את גודל החלון. פונקציות הטיפול בהודעות WM_CREATE ו- WM_PAINT מבצעות את העיבוד המעשי בתוכנית זו.

7.11 רשימה מפורטת של קבצי-על משופרים (Enumerating The Enhanced Metafiles)

התוכניות יכולות להפעיל קבצי-על לצורך שימוש חוזר בהוראות מאוחסנות של ממשק ההתקן הגרפי. למעשה, באפשרותך לאחסן קבוצות רבות של הוראות בקובץ-על נתון. כל קבוצת הוראות היא **רשומה** (Record) בקובץ-העל. יותר מאוחר, יתכן שתרצה לגשת ולהשתמש בקבוצת הוראות מסוימת מתוך קובץ-העל, או רק לבדוק אילו הוראות יש בו.

כדי לעשות זאת, תוכל להשתמש בפונקציה EnumEnhMetaFile כדי להכין רשימה מפורטת (Enumerate) של כל הרשומות שבקובץ-העל. הפונקציה EnumEnhMetaFile מקבלת כל רשומה בקובץ, ומעבירה אותה ל**פונקציית המשוב** (Callback Function) שאתה מגדיר, כדי להכין רשימה מפורטת של הרשומות שבמבנה קובץ-העל המשופר. פונקציית המשוב שמסופקת על ידי היישום מטפלת בכל רשומה כנדרש. הרישום נמשך עד שפונקציית המשוב שמסופקת על ידי היישום מגיעה לרשומה האחרונה, או כאשר הפונקציה מחזירה אפס. את הפונקציה EnumEnhMetaFile כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL EnumEnhMetaFile(
  HDC hdc,                  // handle to device context
  HENHMETAFILE hemf,        // handle to enhanced metafile
  ENHMFENUMPROC lpEnhMetaFunc,
                            // pointer to callback function
  LPVOID lpData,            // pointer to callback function data
  CONST RECT *lpRect        // pointer to bounding rectangle
);
```

הפונקציה EnumEnhMetaFile מקבלת את הפרמטרים שמפורטים בטבלה 7.16.

טבלה 7.16: הפרמטרים של הפונקציה EnumEnhMetaFile.

פרמטר	תיאור
hdc	מזהה הקשר התקן. התוכניות חייבות למסור ידית זו לפונקציית המשוב.
hemf	מזהה קובץ-על משופר.
LpEnhMetaFunc	מצביע לפונקציית המשוב שמסופקת על ידי היישום.
LpData	מצביע לנתוני רשות של פונקציית המשוב.
LpRect	מצביע למבנה מסוג RECT אשר מגדיר את הקואורדינטות של הפינה השמאלית העליונה ואת הפינה הימנית התחתונה של התמונה. מבנה זה מגדיר את מימדי המלבן ביחידות לוגיות.

פונקציית הרישום של API (API Enumeration Function) משתמשת בפונקציית משוב, כדי לעבד את המידע שפונקציית הרישום מחזירה. הפונקציה EnumEnhMetaFile משתמשת בפונקציית משוב עם התבנית הכללית של הפונקציה EnhMetaFileProc. הפונקציה EnhMetaFileProc מעבדת את התבנית המשופרת של קובץ-העל.

את הפונקציה EnhMetaFileProc כותבים בתוכנית כמו בהגדרה שלהלן:

```
int CALLBACK EnhMetaFileProc(
  HDC hDC,                      // handle to device context
  HANDLETABLE FAR *lpHTable,    // pointer to metafile handle
                                // table
  ENHMETARECORD FAR *lpEMFR,    // pointer to metafile record
  int nObj,                     // count of objects
  LPARAM lpData                 // pointer to optional data
);
```

הפונקציה EnhMetaFileProc מקבלת את הפרמטרים שמפורטים בטבלה 7.17.

טבלה 7.17: הפרמטרים של הפונקציה EnhMetaFileProc.

פרמטר	תיאור
hDC	מזהה את הקשר ההתקן שהתוכנית מוסרת לפונקציה EnumEnhMetaFile.
lpHTable	מצביע לטבלה של ידיות הקשורות עם האובייקטים הגרפיים (עטים, מברשות, וכדומה) בקובץ-העל. הכניסה הראשונה מכילה את ידית קובץ-העל המשופר.
LpEMFR	מצביע לאחת הרשומות בקובץ-העל. התוכניות אינן צריכות לשנות רשומה זו (אם יש צורך בשינוי, התוכנית צריכה לבצע אותו על העתק הרשומה).
nObj	מגדיר את מספר האובייקטים עם הידיות הקשורות בטבלת הידיות.
lpData	מצביע לנתונים כלשהם שמסופקים על ידי היישום.

היישום חייב למסור לפונקציה EnumEnhMetaFile את כתובת פונקציית המשוב, כדי שתוכל לרשום אותה. כמו עם שאר פונקציות המשוב שלמדת עליהן בסעיפים קודמים, השם EnhMetaFileProc שומר מקום לשם פונקציית המשוב שמסופקת על ידי התוכנית.

התמונה מכילה נקודות לאורך צלע המלבן שמוצבע על ידי הפרמטר lpRect. אם הפרמטר hDC הוא NULL, Windows מתעלמת מ-lpRect. אם פונקציית המשוב קוראת לפונקציה PlayEnhMetaFileRecord, hDC חייב לזהות הקשר התקן תקף. Windows משתמשת בטרנספורמציות של הקשר ההתקן ושל מצב המיפוי, כדי לבצע את הטרנספורמציה על התמונה שמציגה הפונקציה PlayEnhMetaFileRecord. באפשרותך להשתמש בפונקציה EnumEnhMetaFile כדי לשלב קובץ-על אחד עם קובץ אחר.

כדי להבין יותר טוב את פעולת הפונקציה EnumEnhMetaFile, התבונן בתוכנית **Draw_Shaps** שבתקליטור המצורף לספר זה (בתיקיה Chap07). התוכנית טוענת קובץ-על משופר ומציגה אותו בשטח הלקוח של החלון. כאשר המשתמש בוחר באפשרות !Test, התוכנית מריצה שוב את קובץ-העל, אך הפעם היא משתמשת בפונקציה EnumEnhMetaFile. פונקציית המשוב לוכדת את רשומת קובץ-העל EMR_CREATBRUSHINDIRECT ומחליפה אותה במברשת בצבע אפור בהיר. העיבוד המעשי של התוכנית נעשה בפונקציות WndProc ו-PaintMetaFile. כאשר מהדרים ומפעילים את התוכנית **Draw_Shaps**, התוכנית מציירת תחילה את התמונות עם **רשת** (Cross Hatch), ואחר כך מציירת אותן שוב באפור מלא.

7.12 הפונקציה GetWinMetaFileBits

תוכניות Win32 צריכות להשתמש במבנה קובץ-על משופר. ככל שרוצים שאותה תוכנית תפעל בפלטפורמות רבות, יהיו מקרים שבהם התוכניות תהיינה חייבות להמיר קובץ-על משופר לסגנון קובץ-על של Windows 9x. הפונקציה GetWinMetaFileBits ממירה פורמט של רשומות קובץ-על משופר לפורמט רשומות קובץ-על של Windows, ומאחסנת את הרשומות המומרות בחוצץ המצוין בפרמטר המתאים. את הפונקציה GetWinMetaFileBits כותבים בתוכניות כמו בהגדרה שלהלן:

```
UINT GetWinMetaFileBits(
  HENHMETAFILE hemf,     // handle to the enhanced metafile
  UINT cbBuffer,         // buffer size
  LPBYTE lpbBuffer,      // pointer to buffer
  INT fnMapMode,         // mapping mode
  HDC hdcRef             // handle of reference device context
);
```

הפונקציה GetWinMetaFileBits מקבלת את הפרמטרים שמפורטים בטבלה 7.18.

אם הפונקציה GetWinMetaFileBits מצליחה והמצביע למאגר הוא NULL, הערך המוחזר הוא מספר הבתים שהפונקציה צריכה כדי לאחסן את הרשומות המומרות. אם הפונקציה מצליחה והמצביע למאגר הוא מצביע תקף, הערך המוחזר הוא גודל הנתונים של קובץ-העל, בבתים; אם הפונקציה נכשלת, הערך המוחזר הוא אפס.

הפונקציה GetWinMetaFileBits ממירה קובץ-על משופר לפורמט קובץ-על של Windows, כך שיישום אשר מכיר את הפורמט הישן בלבד יוכל להציג את קובץ-העל. Windows משתמשת **בהקשר ההתקן המיוחס** (Referenced Device Context) כדי לקבוע את הרזולוציה של קובץ-העל המומר. הפונקציה GetWinMetaFileBits אינה מבטלת את תקפות הידית של קובץ-העל המשופר. היישום צריך לקרוא לפונקציה DeleteEnhMetaFile כדי לשחרר את הידית כאשר היישום אינו צריך אותה עוד.

טבלה 7.18: הפרמטרים של הפונקציה GetWinMetaFileBits.

פרמטר	תיאור
hemf	מזהה את קובץ-העל המשופר.
cbBuffer	מגדיר את הגודל, בבתים, של החוצץ שהפונקציה GetWinMetaFileBits מעתיקה לתוכו את הרשומות המומרות.
lpbBuffer	מצביע לחוצץ שהפונקציה GetWinMetaFileBits מעתיקה לתוכו את הרשומות המומרות. אם lpbBuffer הוא NULL, GetWinMetaFileBits מחזירה את מספר הבתים שהפונקציה זקוקה להם, כדי לאחסן את רשומות קובץ-העל המומרות.
fnMapMode	מגדיר את מצב המיפוי שהתוכנית צריכה להשתמש בו עם קובץ-העל המומר.
hdcRef	מזהה את הקשר ההתקן המיוחס.

בגלל ההגבלות לפורמט קובץ-העל של Windows, חלק מהמידע יכול ללכת לאיבוד מתוך קובץ-העל המתקבל. לדוגמה, הפונקציה יכולה אולי להמיר קריאה לפונקציה PolyBezier שנמצאת במקור בקובץ-העל המשופר, בקריאה לפונקציה Polyline שתהיה בקובץ-העל של Windows, מפני שאין פונקציה שקולה ל-PolyBezier בתבנית של קובץ Windows.

כדי להבין יותר טוב את פעולת הפונקציה GetWinMetaFileBits, התבונן בתוכנית **Convert_EMF_WMF** שבתקליטור המצורף לספר זה (בתיקיה Chap07). התוכנית ממירה קובץ-על משופר נתון (במקרה זה, Sample.emf) לקובץ-על סטנדרטי של Windows 9x. התוכנית שומרת אחר כך את קובץ-העל המומר כ- Sample.wmf. העיבוד המעשי של התוכנית נעשה, כרגיל, בפונקציה Wndproc.

7.13 סמלים (Icons)

Windows תומכת בשלושה סוגים בסיסיים של קבצים גרפיים. למדת שבאפשרות התוכניות להשתמש במפת סיביות וקובץ-על כדי לנהל גרפיקה גדולה או קטנה בשטח הלקוח של החלון, ואפילו בחלון עצמו. **סמלים** (Icons) הם למעשה **תת-מחלקה** (Sub-Class) של מפות הסיביות. עם זאת, קבוצת הפעולות אשר תבצע באמצעות סמלים קטנה ומוגבלת, ולכן Windows מתייחסת לסמלים כסוג נפרד של קובץ גרפי.

סמלים הם מפות סיביות קטנות אשר Windows משתמשת בהם כמייצגים חזותיים של אובייקטים, כמו יישומים, קבצים ותיקיות. ב- Windows 9x תראה סמלים רבים ושונים בממשק המשתמש. בגרסאות מוקדמות של Windows או Windows NT, פגשנו בסמלים בעיקר **במנהל היישומים** (Program Manager).

יישום טיפוסי של Windows 9x או Windows NT מכיל לפחות שני סמלים: סמל גדול (32 x 32) וסמל קטן (16 x 16). Windows מציגה את הסמל הקטן בפינה השמאלית העליונה של חלון היישום בשעה שהיישום **מוקטן לסמל** (Minimized, או ממוזער). Windows משתמשת בסמל הגדול עבור סמלי שולחן העבודה ועבור תצוגות סמל גדול.

בדרך כלל יוצרים סמלים עם **עורך התמונות** (Image Editor) של **ערכת פיתוח התוכנה** (Software Development Kit ,SDK) של Windows 9x או עם עורך אחר, כמו למשל עורך הסמלים של Visual C++. אחר כך משתמשים בהוראה ICON כדי להוסיף את הסמלים אל קובץ המשאבים של היישום. דוגמה טיפוסית לשימוש בסמלים היא הרישום של מחלקת החלון הראשי. כמו שראית בתוכניות השונות שלמדת עד כה, התוכניות סמל בשעה שהן רושמות את **מחלקת החלון** (Window Class) על ידי קריאה לפונקציה RegisterClassEx, כמו שרואים להלן:

```
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPTSTR lpCmdLine, int nCmdShow)
 {
   MSG msg;
   HWND hWnd;
   WNDCLASS wc;
   // Register the main application window class.
   //..........................................
   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = (WNDPROC)WndProc;
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInstance;
   wc.hIcon         = LoadIcon(hInstance, lpszAppName);
   wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName;

   if (!RegisterClass(&wc))
      return(FALSE);
// More code here
```

כמו שניתן לראות, קטע הקוד הקודם רושם סמל עם השם שמכיל המצביע למחרוזת lpszAppName, כסמל של החלון הראשי. בסעיפים שיבואו בהמשך, תלמד יותר כיצד להצמיד סמל לתוכנית.

7.14 יצירת סמלים

כמו עם מפות סיביות וקבצי-על, Windows מאפשרת לך ליצור ולשנות סמלים בזמן ריצה. התוכניות ישתמשו בדרך כלל בפונקציה CreateIcon כדי ליצור סמל בזמן ריצה. הפונקציה CreateIcon מאפשרת לתוכניות ליצור סמלים ממערכים בינריים, נתונים של מפות סיביות, ומפות סיביות שאינן תלויות התקן. הפונקציה CreateIcon יוצרת סמל בגודל המוגדר, בצבעים המוגדרים ובתבניות סיביות מוגדרות. את הפונקציה CreateIcon כותבים בתוכניות כמו בהגדרה שלהלן:

```
HICON CreateIcon(
  HINSTANCE hInstance,    // handle to application instance
  int nWidth,             // icon width
  int nHeight,            // icon height
  BYTE cPlanes,           // number of planes in XOR bitmask
  BYTE cBitsPixel,        // number of bits per pixel in XOR
                          // bitmask
  CONST BYTE *lpbANDbits, // pointer to AND bitmask array
  CONST BYTE *lpbXORbits, // pointer to XOR bitmask array
);
```

הפונקציה CreateIcon מקבלת את הפרמטרים שמפורטים בטבלה 7.19.

הפרמטרים nWidth ו-nHeight חייבים להגדיר רוחב וגובה אשר נתמכים על ידי מנהל ההתקן של התצוגה הנוכחית, מפני שהמערכת אינה יכולה ליצור סמלים בעלי גדלים אחרים. כדי לדעת את הרוחב והגובה של הסמלים שנתמכים על ידי מנהל ההתקן של התצוגה, צריך להשתמש בפונקציה GetSystemMetrics, כאשר מגדירים את הערך SM_CXICON או SM_CYICON.

טבלה 7.19: הפרמטרים של הפונקציה CreateIcon.

פרמטר	תיאור
hInstance	מזהה את מופע המודול שיצר את הסמל.
nWidth	מגדיר את רוחב הסמל, בפיקסלים.
nHeight	מגדיר את גובה הסמל, בפיקסלים.
cPlanes	מגדיר את מספר המישורים במסיכת סיביות XOR (XOR bitmask) של הסמל.
cBitsPixel	מגדיר את מספר הסיביות לכל פיקסל במסיכת סיביות XOR של הסמל.
lpbANDbits	מצביע למערך בתים שמכילים את ערכי הסיביות עבור מסיכת סיביות AND (AND Bitmask) של הסמל. מסיכה זו מתארת מפת סיביות בעלת צבע אחד (A Monochrome Bitmap).
lpbXORbits	מצביע למערך של בתים שמכילים את ערכי הסיביות עבור מסיכת סיביות XOR של הסמל. מסיכה זו מתארת מפת סיביות בעלת צבע אחד, או מפת סיביות תלויית התקן בצבע מלא.

הפונקציה CreateIcon יוצרת את הסמל משתי מפות סיביות (אשר הפונקציה משתמשת בהן כמסיכות סיביות); האחת היא מסיכת סיביות AND והשנייה - מסיכת סיביות XOR. מסיכת סיביות AND היא תמיד מפת סיביות מונוכרומטית (בעלת צבע אחד), עם סיבית אחת לכל פיקסל. CreateIcon מייחסת טבלת אמת עבור מסיכות הסיביות הנפרדות ל-AND ול-XOR. תוצאות פעולה זו מוצגות בטבלה 7.20.

טבלה 7.20: טבלת האמת של הפונקציה CreateIcon עבור מסיכות הסיביות AND ו-XOR.

AND bitmask	XOR bitmask	Display (תצוגה)
0	0	Black (שחור)
0	1	White (לבן)
1	0	Screen (מסך)
1	1	Reverse screen (מסך נגטיבי)

כדי להבין יותר טוב את פעולת הפונקציה CreateIcon, התבונן בתוכנית **Create_Icon** שבתקליטור המצורף לספר זה (בתיקיה chap07). התוכנית מגדירה את ערכי הסיבית של שתי המסיכות, מסיכת AND ומסיכת XOR של הסמל, כדי ליצור סמל בעל צבע יחיד (Monochrome Icon). כאשר הפונקציה WndProc מקבלת את ההודעה WM_CREATE, התוכנית יוצרת את הסמל; כאשר הפונקציה WndProc מקבלת את ההודעה WM_PAINT, התוכנית צובעת את הסמל שבתצוגה.

7.15 יצירת סמלים ממשאב

תוכניות יכולות ליצור סמלים במספר דרכים שונות. בדרך כלל הן אינן יוצרות שתי מפות סיביות ושתי מסיכות סיביות בזיכרון, כמו שעשתה התוכנית **Create_Icon** שהוצגה קודם. כמו עם **טבלאות מחרוזות** (String Tables) ומידע אחר המיועדים לשימוש חוזר, אפשר לטעון את מרכיב הסיביות של הסמל מתוך קובץ משאבים ולהפוך את הסיביות לסמל ממשי. כדי לבצע עיבוד כזה, התוכניות משתמשות בפונקציה CreateIconFromResource. פונקציה זו יוצרת סמל או **סמן** (Cursor) מסיביות משאב אשר מתארות את הסמל. את הפונקציה CreateIconFromResource כותבים בתוכניות כמו בהגדרה שלהלן:

```
HICON CreateIconFromResource(
  BYTE presbits,          // pointer to icon or cursor bits
  DWORD dwResSize,        // number of bytes in bit buffer
  BOOL fIcon,             // icon or cursor flag
  DWORD dwVer,            // Windows format version
);
```

הפונקציה CreateIconFromResource מקבלת את הפרמטרים שמפורטים בטבלה 7.21.

טבלה 7.21: הפרמטרים של הפונקציה CreateIconFromResource.

פרמטר	תיאור
presbits	מצביע למאגר, אשר מכיל את משאב הסיביות של הסמל או הסמן. קריאה לפונקציות LookupIconIdFromDirectory (ב- Windows 9x, באפשרותך גם כן לקרוא לפונקציה LookupIconIdFromDirectoryEx) ו-LoadResource טוענת לרוב את הסיביות האלו.
dwResSize	מגדיר את הגודל, בבתים, של קבוצת הסיביות שמוצבעת על ידי הפרמטר Presbits.
fIcon	מגדיר אם הפונקציה צריכה ליצור סמל או סמן. אם פרמטר זה הוא True, הפונקציה יוצרת סמל; ואם הוא False, הפונקציה יוצרת סמן.
dwVer	מגדיר את מספר הגירסה של תבנית הסמל או הסמן עבור משאב הסיביות שמוצבע על ידי הפרמטר Presbits.

הפרמטר dwVer יכול לקבל את אחד הערכים שמפורטים בטבלה 7.22.

טבלה 7.22: ערכי הפרמטר **dwVer**.

פורמט	dwVer
Windows 2.x	0x00020000
Windows 3.x	0x00030000

כל יישומי מיקרוסופט מבוססי Win32 משתמשים בתבנית של Windows 9x עבור סמלים וסמנים. הפונקציות CreateIconFromResource, CreateIconIndirect, GetIconInfo ו-LookupIconIdFromDirectory (Windows 9x, מאפשרת גם לקרוא לפונקציות CreateIconFromResourceEx ו-LookupIconIdFromDirectoryEx) מאפשרות ליישומי מסגרת ודפדפני סמלים לבחון ולהשתמש במשאבים דרך המערכת.

כדי להבין יותר טוב את פעולת הפונקציה CreateIconFromResource, התבונן בתוכנית **Display_Res_Icon** שבתקליטור המצורף לספר זה (בתיקיה chap07). התוכנית **Display_Res_Icon** משתמשת באוסף של פונקציות לניהול משאבים כדי לאתר סמל בקובץ המשאבים, למצוא את המקום שלו בדיסק ולטעון את הסמל לזיכרון. התוכנית מציגה אחר כך את הסמל בשטח הלקוח של החלון. הטיפול המעשי קורה לאחר שהמשתמש בוחר באפשרות !Test. הקוד שמבצע טיפול זה נמצא בפונקציה WndProc.

7.16 הפונקציה CreateIconIndirect

התוכניות יודעות ליצור סמלים ממפות סיביות או ממזהה משאב. באפשרותן גם ליצור סמלים מערך של מבנה. משתמשים בפונקציה CreateIconIndirect כדי ליצור סמלים ממרכיבים שאינך מגדיר אותם בתוכנית, או בקובץ משאבים. הפונקציה יוצרת סמל או סמן ממבנה מסוג ICONINFO. את הפונקציה CreateIconIndirect כותבים בתוכניות כמו בהגדרה שלהלן:

```
HICON CreateIconIndirect(PICONINFO piconinfo);
```

המצביע piconinfo מצביע למבנה מסוג ICONINFO שהפונקציה משתמשת בו כדי ליצור את הסמל או את הסמן. אם הפונקציה מצליחה, הערך המוחזר הוא ידית הסמל או הסמן שיצרה הפונקציה. המערכת מעתיקה את מפות הסיביות שבמבנה ICONINFO לפני יצירת הסמל או הסמן. היישום חייב להמשיך לנהל את מפות הסיביות המקוריות ולמחוק אותן כאשר אין עוד צורך בהן. המבנה ICONINFO מכיל מידע על סמל או סמן.

Windows API מגדיר את המבנה ICONINFO כמו שרואים בדוגמה זו:

```
typedef struct _ICONINFO {
  BOOL     fIcon;
  DWORD    xHotspot;
  DWORD    yHotspot;
  HBITMAP hbmMask;
  HBITMAP hbmColor;
} ICONINFO;
```

טבלה 7.23 מפרטת את איברי המבנה ICONINFO.

טבלה 7.23: איברי המבנה **ICONINFO**.

פרמטר	תיאור
fIcon	מגדיר אם מבנה זה מגדיר סמל או סמן. ערך True מגדיר סמל; False - מגדיר סמן.
xHotspot	מגדיר את קואורדינטת X (X-Coordinate) של **נקודת המגע** (Hot Spot) של הסמן. אם מבנה זה מגדיר סמל, נקודת המגע תמיד נמצאת במרכז הסמל, והפונקציות אשר משתמשות במבנה ICONINFO מתעלמות מהאיבר xHotspot.
yHotspot	מגדיר את קואורדינטת Y (Y-Coordinate) של **נקודת המגע** (Hot Spot) של הסמן. אם מבנה זה מגדיר סמל, נקודת המגע תמיד נמצאת במרכז הסמל, והפונקציות אשר משתמשות במבנה ICONINFO מתעלמות מהאיבר yHotspot.

פרמטר	תיאור
hbmMask	מגדיר את מסיכת סיביות מפת הסיביות של הסמל. אם המבנה מגדיר סמל שחור לבן, מסיכה זו מפורמטת, כך שהחצי העליון הוא מסיכת הסיביות AND של הסמל והחצי התחתון הוא מסיכת הסיביות XOR של הסמל. על פי תנאי זה, הגובה צריך להיות זוגי בכפולות של שתיים. אם מבנה זה מגדיר סמל צבעוני, מסיכה זו מגדירה רק את מסיכת סיביות AND של הסמל.
hbmColor	מזהה את צבע מפת הסיביות של הסמל. איבר זה יכול להיות רשות אם המבנה מגדיר סמל שחור לבן. CreateIconIndirect מספקת את מסיכת סיביות AND של hbmMask עם הדגל SRCAND של היעד. באופן עקבי, CreateIconIndirect משתמשת בדגל SRCINVERT כדי לספק את מפת הסיביות הצבעונית (על ידי השימוש ב- XOR) ליעד.

בקיצור, המבנה ICONINFO מגדיר את מפת הסיביות המונוכרומטית ומפת הסיביות הצבעונית, והפונקציה CreateIconIndirect מחברת את מפות הסיביות בהתבסס על הערכים שבמבנה ICONINFO.

כדי להבין יותר טוב את פעולת הפונקציה CreateIconIndirect, התבונן בתוכנית **Two_Icons** שבתקליטור המצורף לספר זה (בתיקיה Chap07). התוכנית **Two_Icons** מחברת שתי מפות סיביות למפת סיביות שלישית ויוצרת סמל. כרגיל, העיבוד המעשי של התוכנית נמצא בפונקציה WndProc.

7.17 הפונקציה LoadIcon

בסעיפים הקודמים למדת שאפשר להשתמש בתוכניות במספר שיטות שונות כדי ליצור סמלים בזמן ריצה. כמו שראית בתוכניות אחרות, הן משתמשות על פי רוב בפונקציה LoadIcon כדי לטעון לתוכנית סמל מקובץ המשאבים שלה. למדת שהפונקציה LoadIcon מספקת לתוכניות דרך נוחה ויעילה לטעינת סמלים שכבר נוצרו. הפונקציה LoadIcon טוענת את משאב הסמל המוגדר מקובץ ההפעלה (EXE.) שקשור למופע התוכנית. את הפונקציה LoadIcon כותבים בתוכניות כמו בהגדרה שלהלן:

```
HICON LoadIcon(HINSTANCE hInstance, LPCTSTR lpIconName);
```

הפרמטר hInstance מזהה את מופע המודול אשר קובץ ההפעלה שלו מכיל את הסמל שהפונקציה LoadIcon צריכה לטעון. הפרמטר hInstance חייב להיות NULL כאשר הפונקציה LoadIcon צריכה לטעון סמל סטנדרטי (Standard Icon). הפרמטר lpIconName מצביע למחרוזת המסתיימת ב-NULL אשר מכילה את שם משאב הסמל שהפונקציה LoadIcon צריכה לטעון. לחילופין, הפרמטר lpIconName יכול להכיל את מזהה המשאב במלת הסדר-הנמוך ואפס במילת הסדר-הגבוהה. צריך להשתמש במאקרו MAKEINTRESOURCE כדי ליצור ערך מזהה למשאב. כדי להשתמש באחד הסמלים המובנים של Windows, צריך להציב את הערך NULL לפרמטר hInstance ולפרמטר lpIconName - את אחד הערכים שמפורטים בטבלה 7.24.

LoadIcon טוענת את המשאב רק אם התוכנית לא טענה קודם את משאב הסמל; אחרת, LoadIcon מקבלת ידית למשאב הקיים. הפונקציה מחפשת את משאב הסמל שמתאים ביותר לתצוגה הנוכחית. משאב הסמל יכול להיות מפת סיביות צבעונית או מונוכרומטית. LoadIcon יכולה לטעון רק סמל אשר הגודל שלו מתאים לערכי מידות המערכת SM_CXICON ו- SM_CYICON. השתמש בפונקציה LoadImage כדי לטעון סמלים בעלי גודל אחר.

טבלה 7.24: הסמלים המובנים ב-Windows.

ערך	תיאור
IDI_APPLICATION	סמל ברירת המחדל של היישום.
IDI_ASTERISK	סמל כוכבית (Asterisk, שימושי בהודעות מידע).
IDI_EXCLAMATION	סמל סימן קריאה (Exclamation Point, שימושי בהודעות אזהרה).
IDI_HAND	סמל תמרור "עצור" (Hand-Shaped Icon, שימושי בהודעות אזהרה רציניות).
IDI_QUESTION	סמל סימן שאלה (Question Mark, שימושי בהודעות אישור).
IDI_WINLOGO	הלוגו של Windows.

7.18 הפונקציה LoadImage טוענת סוגים גרפיים רבים

למדת שתוכניות יכולות להשתמש בפונקציה LoadIcon כדי לטעון סמל מקובץ המשאבים שלהן. התוכניות גם יכולות להשתמש בפונקציות LoadBitmap ו-LoadCursor כדי לטעון מפות סיביות וסמנים, בהתאמה, מקובץ המשאבים של התוכנית. לחילופין, באפשרות התוכניות להשתמש בפונקציה LoadImage, אשר טוענת סמל, סמן, או מפת סיביות. את הפונקציה LoadImage כותבים בתוכניות כמו בהגדרה שלהלן:

```
HANDLE LoadImage(
  HINSTANCE hinst,          // handle of the instance that
                            // contains the image
  LPCTSTR lpszName,         // name of identifier of image
  UINT uType,               // type of image
  int cxDesired,            // desired width
  int cyDesired,            // desired height
  UINT fuLoad               // load flags
);
```

הפונקציה LoadImage מקבלת את הפרמטרים שמפורטים בטבלה 7.25 .

טבלה 7.25: הפרמטרים של הפונקציה **LoadImage**.

פרמטר	תיאור
hinst	מזהה מופע של המודול, אשר מכיל את התמונה (Image) שהפונקציה LoadImage צריכה לטעון. צריך להציב ערך אפס לפרמטר זה כדי לטעון תמונת OEM.
lpszName	מזהה את התמונה שהפונקציה LoadImage צריכה לטעון. אם הפרמטר hinst אינו NULL והפרמטר fuLoad אינו מכיל את LR_LOADFROMFILE, אז lpszName הוא מצביע למחרוזת המסתיימת ב-NULL ואשר מכילה את שם משאב התמונה שבמודול hinst. עם זאת, אם הערך של hinst הוא NULL ואינך מגדיר את הקבוע LR_LOADFROMFILE, אז מילת הסדר-הנמוך של פרמטר זה חייבת להיות המזהה של תמונת OEM שהפונקציה LoadImage צריכה לטעון. קובץ הכותר Winuser.h מגדיר את **מזהי תמונת OEM** (OEM Image Identifiers), שיש להם **קידומות** (Prefixes, תחיליות) שמפורטות בטבלה 7.26. תחת Windows 9x, אם הפרמטר fuLoad מכיל את הערך LR_LOADFROMFILE, אז lpszName הוא שם הקובץ שמכיל את התמונה. Windows NT אינה תומכת ב- LR_LOADFROMFILE.
uType	מגדיר את סוג התמונה שהפונקציה LoadImage צריכה לטעון. פרמטר זה יכול להיות אחד הערכים שמפורטים בטבלה 7.27 .
cxDesired	מגדיר בפיקסלים את רוחב הסמל או הסמן. אם פרמטר זה שווה לאפס והפרמטר fuLoad שווה ל- LR_DEFAULTSIZE, הפונקציה משתמשת בערכי מידות המערכת SM_CXICON או SM_CXCURSOR כדי לקבוע את הרוחב. אם פרמטר זה שווה לאפס והקריאה לפונקציה אינה משתמשת ב- LR_DEFAULTSIZE, הפונקציה משתמשת ברוחב המשאב הממשי.
cyDesired	מגדיר בפיקסלים את גובה הסמל או הסמן. אם פרמטר זה שווה לאפס והפרמטר fuLoad שווה ל- LR_DEFAULTSIZE, הפונקציה משתמשת בערכי מידות המערכת SM_CYICON או SM_CYCURSOR כדי לקבוע את הגובה. אם פרמטר זה שווה לאפס והקריאה לפונקציה אינה משתמשת ב- LR_DEFAULTSIZE, הפונקציה משתמשת בגובה המשאב הממשי.
fuLoad	קובע איך הפונקציה טוענת את התמונה. מגדיר צירוף של הקבועים שמפורטים בטבלה 7.28.

כמו שניתן לראות בטבלה 7.25 התוכניות יכולה לטעון תמונות OEM. במקרים כאלה, היא גם יכולה להגדיר אחד ממזהי התמונות שמפורטים בטבלה 7.26.

טבלה 7.26: מזהי התמונות (Image Identifiers).

קידומת	פירוש
OBM_	מפות סיביות OEM (OEM bitmap)
OIC_	סמלי OEM (OEM icons)
OCR_	סמני OEM (OEM cursors)

למדת שבאפשרות התוכניות לטעון באמצעות הפונקציה LoadImage תמונה כלשהי מתוך מספר סוגי תמונות. הפרמטר uType מגדיר את סוג התמונה, והוא יכול להיות אחד מהערכים שמפורטים בטבלה 7.27.

טבלה 7.27: סוגי התמונה האפשריים.

ערך	פירוש
IMAGE_BITMAP	טוען מפת סיביות (Loads a bitmap)
IMAGE_CURSOR	טוען סמן (Loads a cursor)
IMAGE_ICON	טוען סמל (Loads an icon)

באפשרותך לטעון את קובץ התמונה עם מספר אפשרויות עבור תצוגת הקובץ. הפרמטר fuLoad חייב להיות אחד או יותר מהערכים שמפורטים בטבלה 7.28.

טבלה 7.28: אפשרויות הטעינה עבור קובץ התמונה.

ערך	פירוש
LR_DEFAULTCOLOR	דגל ברירת המחדל, פירושו "לא LR_MONOCHROME".
LR_CREATEDIBSECTION	כאשר הפרמטר uType מגדיר IMAGE_BITMAP, הוא גורם לפונקציה להחזיר חלק DIB של מפת סיביות במקום **מפת סיביות תואמת** (Compatible Bitmap). דגל זה שימושי לטעינת מפת סיביות מבלי למפות אותה לצבעי התקן התצוגה.

ערך	פירוש
LR_DEFAULTSIZE	משתמש עבור סמנים או סמלים ברוחב או בגובה שמוגדרים על ידי מידות המערכת, אם הקריאה לפונקציה קובעת את הערכים cxDesired או cyDesired לאפס. אם דגל זה אינו מוגדר והקריאה לפונקציה קובעת את הערכים של cxDesired ו- cyDesired לאפס, הפונקציה משתמשת בגודל המשאב הממשי. אם המשאב מכיל מספר תמונות, הפונקציה משתמשת בגודל של התמונה הראשונה.
LR_LOADFROMFILE	טוען את התמונה מהקובץ שמוגדר על ידי הפרמטר lpszName. אם דגל זה אינו מוגדר, lpszName הוא שם המשאב.
LR_LOADMAP3DCOLORS	מחפש בטבלת הצבעים עבור התמונה ומחליף את הצללים (Shades) של האפור בצבע התלת-מימדי המתאים, כמו שרואים בטבלה 7.29.
LR_LOADTRANSPARENT	מקבל את ערך הצבע של הפיקסל הראשון שבתמונה ומחליף את הכניסה המתאימה שבטבלת הצבעים בצבע ברירת המחדל של החלון (COLOR_WINDOW). כל הפיקסלים בתמונה אשר משתמשים בכניסה זאת הופכים להיות בצבע ברירת המחדל של החלון. ערך זה תקף רק לתמונות שיש להן טבלת צבעים מתאימה. אם fuLoad מכיל שני הערכים LR_LOADTRANSPARENT ו-LR_LOADMAP3DCOLORS, אז LR_LOADTRANSPARENT מקבל את העדיפות. עם זאת, הפונקציה LoadImage מחליפה במקרה זה את הכניסה בטבלת הצבעים, וקובעת COLOR_3DFACE במקום COLOR_WINDOW.
LR_MONOCHROME	טוען את התמונה בשחור ולבן.
LR_SHARED	משתף את ידית התמונה, אם תוכנית אחת או יותר טוענות את התמונה פעמים רבות בו-זמנית. אם אינך קובע את LR_SHARED, קריאה שנייה לפונקציה LoadImage עבור אותו משאב טוענת את התמונה שוב ומחזירה ידית שונה. אל תשתמש ב- LR_SHARED עם תמונות שאין להם גדלים סטנדרטיים שיכולים להשתנות אחרי טעינה, או אשר התוכנית שלך טוענת מקובץ.

כאשר טוענים קובץ תמונה כקובץ תלת-מימדי, Windows API ממפה את הצבעים עבורך. Windows מחליפה כל צבע שבעמודה הימנית של הטבלה 7.29 בצבע שבעמודה השמאלית של הטבלה.

טבלה 7.29: הערכיים התלת-המימדים הממופים של הפונקציה **LoadImage**.

צבע	מוחלף ב:
Dk Gray, RGB(128, 128, 128)	COLOR_3DSHADOW
Gray, RGB(192, 192, 192)	COLOR_3DFACE
Lt Gray, RGB(223, 223, 223)	COLOR_3DLIGHT

כמו שצוין בהסברים שניתנו עד כה, התוכניות יכולות להשתמש בפונקציה LoadImage בכל מצב שאפשר להשתמש בו בפונקציות LoadIcon, LoadBitmap או LoadCursor.

פרק **8**

מבט מקרוב על פקדים

נושא הפקדים (Controls) הוצג לראשונה בפרק 5, במסגרת הדיון בתיבות דו-שיח. פרק זה בוחן מספר פקדים נוספים, לרבות תיבות סימון, כפתורי רדיו, תיבות קבוצה, פקדי טקסט סטטיים, ופסי גלילה. כפי שיתברר לך, חלק גדול מהטכניקות, יחול גם על הפקדים שפרק זה.

✍ **הערה:** אם טרם קראת את פרק 5, העוסק בתיבות דו-שיח, עשה זאת עכשיו, כי אנו משתמשים בתיבות דו-שיח כדי להדגים את הפקדים בהם עוסק פרק זה.

תיבת סימון (Check Box) היא **פקד** (Control) המאפשר למשתמש להפעיל אפשרות מסוימת, או להשביתה. תיבת סימון היא תיבה קטנה, העשויה להכיל סימון "°" או להופיע ריקה. לכל תיבת סימון קשורה תווית, המתארת את האפשרות שאותה תיבה מייצגת. אם התיבה מכילה סימן "°", התיבה **מסומנת** (Checked), והאפשרות שהיא מייצגת נמצאת במצב פעיל. אם התיבה ריקה, אזי היא בלתי מסומנת, או **נקיה** (Cleared), לפיכך, האפשרות שהיא מייצגת מושבתת. בדרך כלל תופיע תיבת סימון כחלק מתיבת דו-שיח, והגדרתה תופיע במסגרת ההגדרה של תיבת הדו-שיח בקובץ המשאבים של התוכנית. כדי להוסיף תיבת סימון לתיבת דו-שיח, השתמש בפקודה CHECKBOX, שהגדרתה הכללית היא:

```
CHECKBOX "string", CBID, X, Y, Widht, Height [, Style]
```

בהגדרה זו, מכיל השדה string את הטקסט שיופיע לצד תיבת הסימון. CBID הוא הערך הקשור בתיבת הסימון. הפינה השמאלית-העליונה של תיבת הסימון תמוקם בנקודה X,Y, והתיבה יחד עם הטקסט הנלווה אליה יהיו בגודל שיתקבל ממכפלת הפרמטר Width בפרמטר Height. הפרמטר Style קובע את הסגנון המדויק של תיבת הסימון. אם לא תציין במפורש סגנון מסוים, תוצג תיבת הסימון במתכונת ברירת המחדל, כאשר התכולה של string מוצגת מימין, והמשתמש יכול להגיע אל התיבה בעזרת מקש Tab. כפי שבוודאי ידוע לך בעקבות השימוש בחלונות 9x, תיבות סימון הן בעצם **מפסקים** (Toggles). כל פעם שאתה בוחר תיבת סימון היא משנה את המצב שלה מ**מסומן** ל**לא מסומן**, ולהיפך. פעולה זו לא נעשית בהכרח באופן אוטומטי.

כשאתה משתמש בפקודה CHECKBOX, אתה יוצר למעשה **תיבת סימון ידנית**, שהתוכנית שלך חייבת לנהלה על ידי סימון והסרת הסימון מהתיבה כל פעם שהיא נבחרת (מייד יתברר לך כיצד). חלונות 9x יכולה לבצע עבורך את פונקציית הניהול הזו אם תיצור תיבת סימון אוטומטית. **תיבת סימון אוטומטית** נוצרת בעזרת הפקודה AUTOCHECKBOX, שתבניתה זהה לזו של הפקודה CHECKBOX. כאשר אתה משתמש בתיבת סימון אוטומטית, חלונות מבצעת באופן אוטומטי את שינוי המצב בתיבה (בין מסומן לבין לא-מסומן) כל פעם שהמשתמש בוחר בתיבה.

8.1 תיבות סימון אוטומטיות

לפני שתמשיך, עליך לשנות בקובץ משאבים הבסיסי עבור תיבות הדו-שיח את שני החלקים הבאים: את הפריט, שמכיל כעת פריט נוסף, Status – ואת ההגדרות לתיבת הדו-שיח, כך שתחיל, תיבת סימון ידנית ותיבת סימון אוטומטית. הקלד עתה שני קטעים אלו, במקום הקודמים:

```
DLGBOX MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",                   IDM_EXIT
    END
    MENUITEM "&Test!",                      IDM_TEST
    MENUITEM "&Status",                     ID_STATUS
    POPUP "&Help"
    BEGIN
        MENUITEM "&About My Application...",  IDM_ABOUT
    END
END
TESTDIALOG DIALOG DISCARDABLE  20, 20, 180, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
        WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "Red",IDD_RED,10,10,44,14
    PUSHBUTTON      "Green",IDD_GREEN,75,10,44,14
    PUSHBUTTON      "Cancel",IDCANCEL,9,54,30,13
    PUSHBUTTON      "OK",IDOK,73,54,30,13
    CHECKBOX        "Checkbox 1",IDD_CB1,118,42,59,12
    CONTROL        "Checkbox 2",IDD_CB2,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,118,54,59,12
END
```

עליך גם להוסיף לקובץ הכותר CheckBox1.h, את הערכים הבאים:

```
#define IDD_CB1    1012
#define IDD_CB2    1013
#define ID_STATUS 40001
```

כל פעם שהמשתמש לוחץ בעכבר על תיבת סימון, או בוחר אותה ומקיש על מקש הרווח, נשלחת הודעת WM_COMMAND לפונקציית הדיאלוג ומילת הסדר-הנמוך של wParam מקבלת את הערך המזהה הקשור לאותה תיבת סימון. אם אתה משתמש בתיבת סימון ידנית, יהיה עליך להגיב לפקודה זו על ידי שינוי מצב התיבה. לשם כך, שלח לתיבת הסימון הודעת BM_SETCHECK בעזרת פונקציית API ‏SendDlgItemMessage(). פונקציה זו נדונה בפרק 5. הגדרתה הכללית מובאת כאן פעם נוספת, לנוחותך:

```
LONG SendDlgItemMessage(HWND hdwnd, int ID, UINT IDMsg,
                        WPARAM wParam, LPARAM lParam);
```

כאשר נשלחת הודעת BM_SETCHECK, קובע הערך שמכיל הפרמטר wParam אם התיבה תסומן, או תנוקה. אם wParam מכיל 1, אזי התיבה תסומן. אם הוא מכיל 0, התיבה תנוקה. לפי ברירת המחדל מופיעות כל תיבות הסימון כשהן ריקות. בעת משלוח הודעת BM_SETCHECK, לא נעשה שימוש בפרמטר lParam.

זכור, אם אתה משתמש בתיבת סימון אוטומטית, אזי מצב התיבה ישתנה באופן אוטומטי כל פעם שהיא תיבחר. אינך צריך לשלוח לתיבת סימון אוטומטית הודעת BM_SETCHECK.

תוכל לקבוע מצב של תיבת סימון אם תשלח לה את ההודעה BM_GETCHECK. התיבה תחזיר ערך 1 אם היא מסומנת, וערך 0 אם לא. במקרה זה, יכילו wParam וגם lParam ערך 0.

לפניך התוכנית **CheckBox1** המדגימה תיבת סימון ידנית ותיבה אוטומטית. כדי להבליט בדוגמה שלנו את ההבדלים בין שני הסוגים, מרגע שתיבת סימון ידנית נבחרת, היא נשארת מסומנת תמיד ואין אפשרות לרוקן אותה. בדוגמה שלאחריה, תלמד איך לטפל בתיבות סימון ידניות (לא הבאנו פה את הפונקציות הבאות: WinMain, RegisterWin95 מפני שלא בוצע בהם שום שינוי).

```
#include <windows.h>
#include "CheckBox1.h"

#if defined (WIN32)
        #define IS_WIN32 TRUE
#else
        #define IS_WIN32 FALSE
#endif
```

```
#define IS_NT       IS_WIN32 && (BOOL)(GetVersion() < 0x80000000)
#define IS_WIN32S   IS_WIN32 && (BOOL)(!(IS_NT) &&
                    (LOBYTE(LOWORD(GetVersion()))<4))
#define IS_WIN95    (BOOL)(!(IS_NT) && !(IS_WIN32S)) && IS_WIN32

HINSTANCE hInst;           // current instance
int status1=0, status2=0; /* holds status of check boxes */

LPCTSTR lpszAppName  = "DlgBox";
LPCTSTR lpszTitle    = "Chap08";

BOOL RegisterWin95( CONST WNDCLASS* lpwc );

int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                  hPrevInstance, LPTSTR lpCmdLine, int
nCmdShow)
{
   MSG      msg;
   HWND     hWnd;
   WNDCLASS wc;

   // Register the main application window class.
   //...........................................
   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = (WNDPROC)WndProc;
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInstance;
   wc.hIcon         = LoadIcon( hInstance, lpszAppName );
   wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName;

   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );
```

```
   hInst = hInstance;
   // Create the main application window.
   //.....................................
   hWnd = CreateWindow( lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                      );
   if ( !hWnd )
      return( FALSE );

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );

   while( GetMessage( &msg, NULL, 0, 0) )
   {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }

   return( msg.wParam );
}


BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style         = lpwc->style;
   wcex.lpfnWndProc   = lpwc->lpfnWndProc;
   wcex.cbClsExtra    = lpwc->cbClsExtra;
   wcex.cbWndExtra    = lpwc->cbWndExtra;
   wcex.hInstance     = lpwc->hInstance;
   wcex.hIcon         = lpwc->hIcon;
   wcex.hCursor       = lpwc->hCursor;
```

```
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;

   // Added elements for Windows 95.
   //..............................
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");

   return RegisterClassEx( &wcex );
}

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
LPARAM lParam )
{
   char str[140];
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                    DialogBox( hInst, "TestDialog", hWnd,
                             (DLGPROC)TestDlgProc );
                    break;

                 case ID_STATUS: /* show check box status */
                    if(status1)
                       strcpy(str, "Checkbox 1 is checked\n");
                    else
                       strcpy(str, "Checkbox 1 is not
                              checked\n");
                    if(status2)
                       strcat(str, "Checkbox 2 is checked");
                    else
                       strcat(str, "Checkbox 2 is not checked");
                       MessageBox(hWnd, str, "Status", MB_OK);
                    break;
```

```
                case IDM_EXIT :
                     DestroyWindow( hWnd );
                     break;
              }
              break;

      case WM_DESTROY :
              PostQuitMessage(0);
              break;

      default :
            return( DefWindowProc( hWnd, uMsg, wParam, lParam));
   }

   return( 0L );
}

LRESULT CALLBACK TestDlgProc( HWND hDlg, UINT uMsg,
                              WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {

      case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
               case IDOK:
                    /* update global checkbox status variables*/
                    status1 = SendDlgItemMessage(hDlg, IDD_CB1,
                       BM_GETCHECK, 0, 0);  // is box checked?
                    status2 = SendDlgItemMessage(hDlg, IDD_CB2,
                       BM_GETCHECK, 0, 0);  // is box checked?
                    EndDialog(hDlg, 0);
                    break;
               case IDD_CB1:
                   /* user selected 1st check box, so check it*/
                    SendDlgItemMessage(hDlg, IDD_CB1,
                                   BM_SETCHECK, 1, 0);
                    break;
```

```
                case IDCANCEL :
                     EndDialog( hDlg, IDCANCEL );
                     break;
                case IDD_RED:
                     MessageBox(hDlg, "You Picked Red", "RED",
                                MB_OK);
                     break;
                case IDD_GREEN:
                     MessageBox(hDlg, "You Picked Green", "GREEN",
                                MB_OK);
                     break;
                                 }
            break;

      default :
            return( FALSE );
   }

   return( TRUE );
}

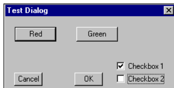
LRESULT CALLBACK About( HWND hDlg,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
   switch (message)
   {
       case WM_INITDIALOG:
          return (TRUE);

      case WM_COMMAND:
      switch(LOWORD(wParam)) {
        case IDCANCEL:
          EndDialog(hDlg, 0);
          return 1;
      }
   }

   return (FALSE);
}
```

תוכנית זו מכילה שני משתנים גלובליים, status1 ו-status2, המכילים את המצב של שתי תיבות הסימון. משתנים אלה מקבלים את הגדרתם עם הלחיצה על לחצן OK בתוך תיבת הדו-שיח. כדי להגדיר מצב של תיבות סימון, בחר באפשרות Dialog בתפריט הראשי. כדי לברר מצב של תיבות סימון, בחר באפשרות Status מהתפריט הראשי (האפשרות Help כלולה כאן רק כמראה מקום).

כשתריץ את התוכנית ותבחר באפשרות Dialog מתוך התפריט הראשי, תופיע לפניך תיבת הדו-שיח הנראית בתרשים 8.1.



תרשים 8.1: תיבת דו-שיח עם תיבת סימון לדוגמה

כיצד לנהל תיבות סימון

התוכנית ליצירת תיבות סימון, כשלעצמה, סובלת משני ליקויים חמורים. ראשית, מצבה של כל תיבת סימון נקבע מחדש בכל פעם שתיבת הדו-שיח מוצגת. כלומר, המצב הקודם של כל תיבה אינו נשמר. שנית, ניתן לסמן את תיבת הסימון הידנית, אך לא ניתן להסיר ממנה את הסימון. כלומר, תיבת הסימון הידנית אינה מפסק (Toggle) במובן האמיתי של המילה, ואינה מתפקדת כמצופה מתיבת סימון. בסעיף זה, נלמד כיצד לטפל באופן יעיל יותר בתיבות סימון.

כיצד להעניק לתיבת סימון תכונות של מפסק

יש לציין, שאמנם קל ופשוט יותר להשתמש בתיבות סימון אוטומטיות, אך אפשר בהחלט להפעיל תיבת סימון במתכונת של מפסק על ידי טיפול נכון בתיבת סימון ידנית. כלומר, התוכנית תצטרך לבצע את כל הפונקציות הניהוליות בעצמה, במקום להניח לחלונות 9x לטפל בעניין. לשם כך, התוכנית תברר תחילה מה מצבה הנוכחי של תיבת הסימון, ולאחר מכן תכתיב לה את המצב ההפוך. הדבר נעשה על ידי הכנסת השינויים הבאים לפונקציית הדיאלוג (קובץ הכולל שינוי זה קיים בתקליטור, ושמו הוא: **CheckBox1_a** . כדי להשתמש בו, החלף את CheckBox1 בקובץ הזה):

```
case IDD_CB1: /* This is a manually managed check box. */
   /* user selected 1st check box, so change its state */
   if(!SendDlgItemMessage(hDlg, IDD_CB1, BM_GETCHECK, 0, 0))
   SendDlgItemMessage(hDlg, IDD_CB1, BM_SETCHECK, 1, 0);
      else /* turn it off */
   SendDlgItemMessage(hDlg, IDD_CB1, BM_SETCHECK, 0, 0);
 break;
```

כיצד לאתחל תיבת סימון

כאמור לעיל, יופיעו תיבות סימון ידניות ואוטומטיות כאחת במצב ריק (כלומר, לא מסומן) כל פעם שתופעל תיבת הדו-שיח המכילה אותן. דבר זה נוח ורצוי בנסיבות מסוימות, אך אין זה מה שהמשתמש מצפה לו כרגיל. ככלל, תיבות סימון מופיעות במצב הקודם שהותכב להן כל פעם שתיבת הדו-שיח מוצגת. אם אתה מעוניין שתיבות הסימון ישקפו את המצב הקודם שלהן, עליך לאתחל אותן כל פעם שתיבת הדו-שיח מופעלת. הדרך הפשוטה ביותר לעשות זאת היא לשלוח להן הודעות BM_SETCHECK מתאימות כאשר תיבת הדו-שיח נוצרת. זכור, כל פעם שתיבת דו-שיח מופעלת, נשלחת אליה הודעת WM_INITDIALOG. כאשר ההודעה מתקבלת, אפשר לקבוע את מצב תיבות הסימון (וכל דבר אחר) בתיבת הדו-שיח.

קטע הקוד הבא מאתחל את תיבות הסימון:

```
case WM_INITDIALOG:
  /* The dialog box has just been displayed.  Set
     the check boxes appropriately. */
  SendDlgItemMessage(hDlg, IDD_CB1, BM_SETCHECK, status1, 0);
  SendDlgItemMessage(hDlg, IDD_CB2, BM_SETCHECK, status2, 0);
  return 1;
```

לפניך התוכנית השלמה **CheckBox2**, הכוללת אתחול תיבות הסימון ומנהלת את תיבת הסימון הידנית. השווה את פעולתה לזו של התוכנית לדוגמה הקודמת. כפי שניתן לצפות, התנהגותה דומה עכשיו לזו של רוב היישומים השכיחים של חלונות (התוכנית נמצאת בתקליטור בתיקיה **CheckBox2**\Chap08).

8.2 הוספת פקדים סטטיים

פקד סטטי (Static Control) הוא אמצעי שאינו מפיק או מקבל הודעות. בקצרה, המונח **פקד סטטי** אינו אלא תיאור של פריט שתפקידו מתמצה בכך שהוא מוצג בתוך תיבת דו-שיח, למשל הודעה מילולית או תיבה המכילה סוגי פקד אחרים. בסעיף זה נבחן שני פקדים סטטיים, **תיבת הטקסט הממורכזת** (Centered Text Box) ו-**תיבת הקבוצה** (Group Box). שני האמצעים האלה נכללים בהגדרת תיבת הדו-שיח שבקובץ המשאבים של התוכנית, בעזרת התוכניות CTEXT ו-GROUPBOX, בהתאמה.

הפקודה CTEXT מפיקה מחרוזת המופיעה כשהיא ממורכזת באזור שהוגדר מראש. לפניך ההגדרה הכללית של הפקודה CTEXT:

```
CTEXT "text", CTID, X, Y, Width, Height [, Style]
```

בדוגמה, text מכיל את המלל שיוצג. CTID הוא הערך הקשור בטקסט זה. הטקסט יוצג בתוך תיבה שפינתה השמאלית-העליונה היא בנקודה X,Y, וגודלה הוא המכפלה של הפרמטר Width בפרמטר Height. הפרמטר Style קובע את הסגנון המדויק של תיבת טקסט. אם לא נבחר שום סגנון מיוחד, יופיע הטקסט בתיבה במתכונת ברירת המחדל. כלומר, המלל ב-text יוצג כשהוא ממורכז בתיבה. תיבת הטקסט עצמה **אינה** מוצגת. היא רק מגדירה את השטח שמותר למלל לתפוס.

הפקודה GROUPBOX מציירת תיבה. בדרך כלל התיבה משמשת כדי לתחום חזותית סוגי פקד אחרים, והיא עשויה להכיל גם כותרת עבור הקבוצה. הגדרתה הכללית של הפקודה GROUPBOX מובאת להלן:

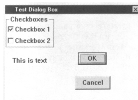
```
GROUPBOX "title", GBID, X, Y, Width, Height [, Style]
```

במקרה זה, title הוא הכותרת של התיבה. CTID הוא הערך הקשור עם המלל שיופיע בתיבה. הפינה השמאלית-העליונה תמוקם בנקודה X,Y, וגודל התיבה יהיה המכפלה של הפרמטר Width בפרמטר Height. הפרמטר Style יקבע בדיוק את הסגנון של תיבת הקבוצה. בדרך כלל אפשר להסתפק בהגדרות ברירת המחדל.

כדי לראות את השפעת שני הפקדים הסטטיים האלה, הוסף את ההגדרות הבאות לתיבת הדו-שיח בקובץ המשאבים שיצרת עבור הדוגמאות הקודמות:

```
GROUPBOX "Checkboxes", ID_GB1, 1, 1, 51, 34
CTEXT "This is text", ID_CT1, 1, 44, 50, 24
```

לאחר שהוספת את השורות, הדר מחדש את הדוגמה הקודמת, הרץ את התוכנית ובחר באפשרות Dialog מתוך התפריט הראשי. תיבת הדו-שיח תיראה עכשיו כמו זו המופיעה בתרשים 8.2. זכור, הפקדים הסטטיים מעניקים חזות שונה לתיבת הדו-שיח, אך אין הם משנים את תפקודה.



תרשים 8.2: הוספת סוגי פקד סטטיים לתיבת דו-שיח

8.3 כפתורי רדיו

הפקד הבא שנבחן הוא **כפתור הרדיו** (Radio Button). כפתורי רדיו משמשים לייצוג אפשרויות שאינן יכולות להתקיים יחד (Mutually Exclusive). כפתור רדיו כולל תווית, המופיעה בצד כפתור קטן. אם הכפתור ריק, האפשרות שהוא מייצג אינה פעילה. אם הכפתור מלא, אזי האפשרות שהוא מייצג נמצאת במצב פעיל. חלונות 9x תומכת בשני סוגים של כפתורי רדיו: ידניים ואוטומטיים. בדומה לתיבת הסימון הידנית, כפתור הרדיו הידני מחייב אותך לבצע את כל הפונקציות הניהוליות. כפתור רדיו אוטומטי מבצע עבורך את כל הפונקציות הניהוליות. הטיפול בכפתורי רדיו מורכב יותר מהטיפול בתיבות סימון. בדרך כלל, ביישומים מופיעים כפתורי רדיו אוטומטיים; על כן נבחן כאן רק סוג זה של כפתורים.

בדומה לפקדים האחרים, כפתורי רדיו אוטומטיים מוגדרים בתוך קובץ המשאבים של התוכנית, במסגרת של הגדרת תיבת הדו-שיח. כדי ליצור כפתור רדיו אוטומטי, השתמש בפקודה AUTORADIOBUTTON, שהגדרתה הכללית מובאת להלן:

```
AUTORADIOBUTTON "string", RBID, X, Y, Width, Height [, Style]
```

בהגדרה זו, מכיל string את המלל שיוצג בצד הכפתור. RBID הוא הערך הקשור בכפתור הרדיו. הפינה השמאלית-העליונה של הכפתור תמוקם בנקודה X,Y, והכפתור יחד עם הטקסט הצמוד אליו, יתפוס שטח שגודלו ייקבע על ידי הכפלת הפרמטר Width בפרמטר Height. הפרמטר Style קובע בדיוק את הסגנון של כפתור הרדיו. אם לא צוין שום סגנון מיוחד, יופיע הכפתור במתכונת ברירת המחדל, כאשר המלל ב-string מוצג מימין לכפתור, והמשתמש יכול להגיע אליו על ידי הקשה על מקש Tab.

בדרך כלל, כפתורי רדיו משמשים ליצירת קבוצות של אפשרויות שאינן יכולות להתקיים יחד, אלא רק אחת בלבד (Mutually Exclusive). כאשר אתה משתמש בכפתורי רדיו אוטומטיים ליצירת קבוצה כזו, חלונות 9x מנהלת את הכפתורים באופן אוטומטי במתכונת של "אחד בלבד". כלומר, בחירה בכפתור מסוים מבטלת את הבחירה בכפתור הקודם. **לא** ניתן לבחור ביותר מכפתור אחד בכל פעם.

תוכל להכתיב לכפתורי רדיו (גם אוטומטיים) מצב ידוע, אם תשלח להם את הודעת BM_SETCHECK באמצעות פונקציית API בשם ()SendDlgItemMessage. הערך של wParam יקבע אם הכפתור יהיה במצב פעיל, או מושבת (מסומן, או ריק). אם wParam מכיל 1, אזי הכפתור יהיה מסומן. אם יכיל 0, הכפתור יופיע ריק. לפי ברירת המחדל, מופיעים כל הכפתורים כשהם ריקים.

✍ **הערה:** גם אם תשתמש בכפתורי רדיו אוטומטיים, ניתן להפעיל ידנית יותר מאפשרות אחת, או להסיר את הסימון מכל האפשרויות באמצעות הפונקציה ()SendDlgItemMessage. אולם הסגנון החלונאי המקובל מכתיב שימוש בכפתורי רדיו במתכונת של "אחד בלבד", כאשר כל כפתור פעיל מבטל את כל השאר. אנו ממליצים מאוד שתקפיד לנהוג לפי כלל זה.

תוכל לברר מצב של כפתור רדיו באמצעות משלוח הודעת BM_GETCHECK. הכפתור יחזיר ערך 1 אם הוא פעיל, וערך 0 אם לא.

כדי להוסיף כפתורי רדיו לַתוכנית לדוגמה, הוסף תחילה את השורות הבאות להגדרת תיבת הדו-שיח שבקובץ המשאבים. שים לב שתיבת קבוצה מתוספת כאן, ומכילה את כפתורי הרדיו. הדבר אינו הכרחי כמובן, אך הקבצות כאלה הן כלי נפוץ בעיצוב תיבות דו-שיח.

```
AUTORADIOBUTTON "Radio 1", ID_RB1, 60, 10, 48, 12
AUTORADIOBUTTON "Radio 2", ID_RB2, 60, 22, 48, 12
GROUPBOX "Radio Group", ID_GB2, 58, 1, 51, 34
```

לפניך התוכנית הקודמת, במתכונת מורחבת הכוללת גם שני כפתורי רדיו. כפתורי הרדיו הם מסוג אוטומטי, ולכן אין תוספות רבות לתוכנית. שים לב שמצב הכפתורים נשמר בשני המשתנים הגלובליים rbstatus1 ו-rbstatus2. הערכים בשני המשתנים האלה משמשים לקביעת המצבים ההתחליים של הכפתורים ולהצגת מצב הכפתורים בתגובה לבחירה באפשרות Status מהתפריט הראשי.

כשתריץ את התוכנית **CheckBox3**, תיבת הדו-שיח תראה כמו זו בתרשים 8.3.

```
#include <windows.h>
#include "CheckBox3.h"

#if defined (WIN32)
                        #define IS_WIN32 TRUE
#else
                        #define IS_WIN32 FALSE
#endif

#define IS_NT       IS_WIN32 && (BOOL)(GetVersion() < 0x80000000)
#define IS_WIN32S  IS_WIN32 && (BOOL)(!(IS_NT) &&
                   (LOBYTE(LOWORD(GetVersion()))<4))
#define IS_WIN95   (BOOL)(!(IS_NT) && !(IS_WIN32S)) && IS_WIN32

HINSTANCE hInst;            // current instance
int cbstatus1=0, cbstatus2=0; /* holds status of check boxes */
int rbstatus1=0, rbstatus2=0; /* holds status of check boxes */

LPCTSTR lpszAppName  = "DlgBox";
LPCTSTR lpszTitle    = "Chap08";

BOOL RegisterWin95( CONST WNDCLASS* lpwc );
extern LRESULT CALLBACK DialogFunc(HWND hdwnd, UINT message,
                              WPARAM wParam, LPARAM lParam);

int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                   hPrevInstance, LPTSTR lpCmdLine, int
nCmdShow)
{
   MSG      msg;
   HWND     hWnd;
   WNDCLASS wc;

   // Register the main application window class.
```

```
//..........................................
wc.style         = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc   = (WNDPROC)WndProc;
wc.cbClsExtra    = 0;
wc.cbWndExtra    = 0;
wc.hInstance     = hInstance;
wc.hIcon         = LoadIcon( hInstance, lpszAppName );
wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wc.lpszMenuName  = lpszAppName;
wc.lpszClassName = lpszAppName;

if ( IS_WIN95 )
{
   if ( !RegisterWin95( &wc ) )
      return( FALSE );
}
else if ( !RegisterClass( &wc ) )
   return( FALSE );

hInst = hInstance;

// Create the main application window.
//...................................
hWnd = CreateWindow( lpszAppName,
                     lpszTitle,
                     WS_OVERLAPPEDWINDOW,
                     CW_USEDEFAULT, 0,
                     CW_USEDEFAULT, 0,
                     NULL,
                     NULL,
                     hInstance,
                     NULL
                   );

if ( !hWnd )
   return( FALSE );

ShowWindow( hWnd, nCmdShow );
UpdateWindow( hWnd );
```

```
   while( GetMessage( &msg, NULL, 0, 0) )
   {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }

   return( msg.wParam );
}

BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style         = lpwc->style;
   wcex.lpfnWndProc   = lpwc->lpfnWndProc;
   wcex.cbClsExtra    = lpwc->cbClsExtra;
   wcex.cbWndExtra    = lpwc->cbWndExtra;
   wcex.hInstance     = lpwc->hInstance;
   wcex.hIcon         = lpwc->hIcon;
   wcex.hCursor       = lpwc->hCursor;
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;

   // Added elements for Windows 95.
   //..............................
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");

   return RegisterClassEx( &wcex );
}

/* This function is called by Windows 95 and is passed
   messages from the message queue.  */
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
                         WPARAM wParam, LPARAM lParam)
```

```
{
  char str[255];

  switch(message) {
    case WM_COMMAND:
      switch(LOWORD(wParam)) {
        case IDM_TEST:
          DialogBox(hInst, "TESTDIALOG", hWnd,
(DLGPROC)DialogFunc);
          break;
        case ID_STATUS:
          if(cbstatus1) strcpy(str, "Checkbox 1 is checked\n");
             else strcpy(str, "Checkbox 1 is not checked\n");
          if(cbstatus2) strcat(str, "Checkbox 2 is checked\n");
             else strcat(str, "Checkbox 2 is not checked\n");
          if(rbstatus1) strcat(str, "Radio 1 is checked\n");
             else strcat(str, "Radio 1 is not checked\n");
          if(rbstatus2) strcat(str, "Radio 2 is checked");
             else strcat(str, "Radio 2 is not checked");
          MessageBox(hWnd, str, "", MB_OK);
          break;
      }
      break;
    case WM_DESTROY: /* terminate the program */
      PostQuitMessage(0);
      break;
    default:
      /* Let Windows 95 process any messages not specified in
      the preceding switch statement. */
      return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}

/* A simple dialog function. */
LRESULT CALLBACK DialogFunc(HWND hdwnd, UINT message,
                            WPARAM wParam, LPARAM lParam)
{
  switch(message) {
```

```
    case WM_INITDIALOG:
      /* The dialog box has just been displayed.  Set
         the check boxes and radio buttons appropriately. */
      SendDlgItemMessage(hdwnd, ID_CB1, BM_SETCHECK, cbstatus1, 0);
      SendDlgItemMessage(hdwnd, ID_CB2, BM_SETCHECK, cbstatus2, 0);
      SendDlgItemMessage(hdwnd, ID_RB1, BM_SETCHECK, rbstatus1, 0);
      SendDlgItemMessage(hdwnd, ID_RB2, BM_SETCHECK, rbstatus2, 0);
      return 1;
    case WM_COMMAND:
      switch(LOWORD(wParam)) {
        case IDCANCEL:
          EndDialog(hdwnd, 0);
          return 1;
        case IDOK:
          /* update global check box status variables */
          cbstatus1 = SendDlgItemMessage(hdwnd, ID_CB1,
             BM_GETCHECK, 0, 0);  // is box checked?
          cbstatus2 = SendDlgItemMessage(hdwnd, ID_CB2,
             BM_GETCHECK, 0, 0);  // is box checked?

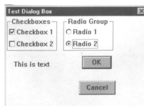
          /* now, update global check box status variables */
          rbstatus1 = SendDlgItemMessage(hdwnd, ID_RB1,
             BM_GETCHECK, 0, 0);  // is button checked?
          rbstatus2 = SendDlgItemMessage(hdwnd, ID_RB2,
             BM_GETCHECK, 0, 0);  // is button checked?

          EndDialog(hdwnd, 0);
          return 1;
      case ID_CB1: /* This is a manually managed check box. */
        /* user selected 1st check box, so change its state */
          if(!SendDlgItemMessage(hdwnd, ID_CB1, BM_GETCHECK, 0, 0))
         SendDlgItemMessage(hdwnd, ID_CB1, BM_SETCHECK, 1, 0);
          else /* turn it off */
         SendDlgItemMessage(hdwnd, ID_CB1, BM_SETCHECK, 0, 0);
          return 1;
      }
  }
  return 0;
}
```

```
LRESULT CALLBACK About( HWND hDlg,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
   switch (message)
   {
      case WM_INITDIALOG:
         return (TRUE);

      case WM_COMMAND:
      switch(LOWORD(wParam)) {
        case IDCANCEL:
          EndDialog(hDlg, 0);
          return 1;
      }
   }

   return (FALSE);
}
```

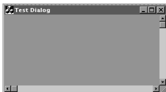


תרשים 8.3: תיבת דו-שיח עם כפתורי רדיו

פרק 9

הקדמה לפסי גלילה (Scroll Bars)

כמו שלמדת, חלון יישום יכול להציג דברים אחדים. מפעם לפעם החלון מציג **אובייקט נתונים** (Data Object), כמו מסמך או צורה גרפית, שגדולה משטח הלקוח של החלון. כאשר יש לחלון **פס גלילה** (Scroll Bar), באפשרות המשתמש לגלול את אובייקט הנתונים שבתוך שטח הלקוח כדי לראות את כל המסמך או את כל הצורה הגרפית. חלון היישום צריך לכלול פס גלילה כאשר התכולה של שטח הלקוח גדולה ברוחב או באורך מגודל שטח הלקוח של החלון. בגלילה סטנדרטית יש שלושה מרכיבים: **החיצים** משני הצדדים, רוחב **פס הגלילה** או האורך שלו, **והגררה** (Thumb) של פס הגלילה (זו התיבה הקטנה שהמשתמש גורר לאורך פס הגלילה). לדוגמה, תרשים 9.1 מציג חלון שבו פסי גלילה צמודים לשפות שטח הלקוח של החלון, בקצותיהם יש חיצים ובתוכם - גררה.



תרשים 9.1: חלון עם פסי גלילה פנימיים והמרכיבים שלהם.

9.1 סוגי פסי הגלילה

למדת בסעיף הקודם שלכל פס גלילה סטנדרטי (Standard Scroll Bar), ללא קשר במקומו ובכיוונו, יש תכונות כלליות המשותפות לו עם פסי גלילה אחרים. למעשה, יש שתי קטגוריות של פסי גלילה. הקטגוריה הראשונה היא **פסי גלילה בשטח הלקוח** (Client-Area Scroll Bars). פסי גלילה אלה יכולים להיות אנכיים או אופקיים, ואולי יופיעו כפסי גלילה סטנדרטיים או **כפסי עקיבה** (Track Bar). הקטגוריה השנייה היא **פסי גלילה שאינם בשטח הלקוח** (Non-Client-Area Scroll Bars). Windows מצמידה את פסי הגלילה שאינם בשטח הלקוח לגבולות החלון.

9.2 הוספת פסי גרירה לחלון

הוספת פס גרירה מחוץ לשטח הלקוח

באפשרותך ליצור ביישומים שלך פסי גלילה שאינם בשטח הלקוח, באמצעות המבנה WNDCLASS שהתוכנית מוסרת לפונקציה CreateWindow. כאשר מצמידים פס גלילה לשפת החלון, Windows מחסרת באופן אוטומטי את רוחב פס הגלילה מתוך שטח הלקוח של החלון, כך שציור בשטח הלקוח לא יוצג אף פעם על פסי הגלילה. באפשרות התוכנית גם לקרוא לפונקציה ShowScrollbar כדי להצמיד את פסי הגלילה לשפה הפנימית של החלון.

בנוסף להצמדת פס הגלילה לשפה הפנימית של החלונות, מערכת ההפעלה מצמידה באופן אוטומטי פסי גלילה ל**תיבות רשימה** (List Boxes) ול**תיבות משולבות** (Combo Boxes) כאשר רשימת הפריטים גולשת מעבר לגודלו של חלון הרשימה. באפשרות התוכניות גם להצמיד פסי גלילה ל**תיבות עריכה** (Edit Boxes). עם זאת, תיבות עריכה בעלות **שורה אחת** (Single-Line) תומכות רק בפס גלילה אופקי, בשעה שתיבות עריכה **מרובות שורות** (Multiple Line) מאפשרות תמיכה בשני פסי גלילה: אופקי ואנכי.

הוספת פס גרירה כסוג פקד לתיבת דו-שיח

פקד פס הגלילה מחייב שימוש בכמה טכניקות מיוחדות.

כדי להוסיף פקד פס גלילה כסוג פקד לתיבת דו-שיח, השתמש במשפט SCROLLBAR, שהגדרתו הכללית היא:

```
SCROLLBAR SBID, X, Y, Width, Height [, Style]
```

בהגדרה זו, SBID הוא הערך הקשור עם פס הגלילה. הפינה השמאלית-העליונה של פס הגלילה תמוקם בנקודה X,Y, וגודלו יהיה מכפלת הפרמטר Width בפרמטר Height. הפרמטר Style יקבע במדויק את סגנון פס הגלילה. לפי ברירת המחדל, הסגנון הוא SBS_HORZ, היוצר פס גלילה אופקי. כדי ליצור פס גלילה אנכי, ציין SBS_VERT עבור הפרמטר Style. אם אתה רוצה בפס גלילה שיגיב להוראות מהמקלדת, כלול גם את הסגנון WS_TABSTOP.

למשל, השורה הבאה יוצרת פס גלילה אנכי:

```
SCROLLBAR ID_SB1, 130, 10, 10, 70, SBS_VERT ¦ WS_TABSTOP
```

9.3 קבלת הודעות מפסי גלילה

שלא כמו שאר הפקדים, פסי גלילה המתפקדים כפקדים אינם מפיקים הודעות WM_COMMAND. במקום זה, פסי גלילה, בין אם הם פקדים ובין אם הם פסי גלילה של חלון, שולחים הודעות WM_VSCROLL או WM_HSCROLL אל היישום, על פי פס הגלילה שעליו לוחצים, האופקי או האנכי. מילת הסדר-הנמוך של הפרמטר lParam מודיעה לפונקציה המטפלת היכן היה העכבר כאשר המשתמש לחץ על הלחצן שלו. כתלות במקום שבו היה העכבר בזמן של הלחיצה, Windows שולחת אחת מעשר הודעות אל התוכנית. תרשים 9.2 מראה את המקומות האפשריים ללחיצה בעכבר ואת ההודעה המתאימה שמקבל החלון.



SB_THUMBPOSITION (Released)
SB_THUMBTRACK (Pressed)

תרשים 9.2: חלק מההודעות ש-Windows יוצרת כתוצאה מלחיצה בעכבר על פסי הגלילה.

כאשר המשתמש משחרר את לחצן העכבר לאחר לחיצה במקום כלשהו בפס הגלילה, Windows שולחת את ההודעה WM_ENDSCROLL אל היישום. אם המשתמש הזיז את הגררה בעזרת העכבר, Windows יוצרת את ההודעה SB_THUMBPOSITION כאשר המשתמש משחרר את לחצן העכבר.

9.4 קביעת הטווח של פס הגלילה (SetScrollRange)

לפני שתוכל להשתמש בפס גלילה כפקד, עליך להגדיר את הטווח שלו. הטווח של פס גלילה קובע כמה מצבים יתקיימו בין שני קצות פס הגלילה. לפי ברירת המחדל, הטווח של פסי גלילה לחלונות הוא 0 עד 100. לפסי גלילה המתפקדים כפקדים יש טווח ברירת מחדל של 0 עד 0, לכן, יש להגדיר את הטווח לפני שניתן יהיה להשתמש בפס הגלילה כפקד. כדי לקבוע את הטווח של פס גלילה, השתמש בפונקציה SetScrollRange(), שהגדרתה היא:

```
BOOL SetScrollRange(HWND hwnd, int which, int min,
                              int max, BOOL repaint);
```

במקרה זה, hwnd היא הידית המגדירה את פס הגלילה. בפסי גלילה של חלונות, זוהי ידית החלון; בפסי גלילה המתפקדים כפקדים, זוהי הידית של פס הגלילה (זכור שפסי גלילה שהם פקדים, מעבירים את הידית שלהם ב- lParam). הערך של which יקבע איזה פס גלילה עומד לקבל הגדרת טווח. אם אתה עוסק בקביעת הטווח של פס הגלילה האנכי של חלון מסוים, חייב פרמטר זה להכיל את הערך SB_VERT. אם אתה עוסק בקביעת הטווח של פס גלילה אופקי של חלון נתון, חובה לרשום את הערך SB_HORZ.

כדי להגדיר את הטווח של פס גלילה שהוא מסוג פקד, עליך לרשום את הערך SB_CTL. הערכים בשדות min ו- max קובעים את הטווח. הערכים המותרים הם בין 0 ל-32,767. אם repaint מכיל ערך true, אזי פס הגלילה יצויר מחדש לאחר קביעת הטווח. אם הוא מכיל ערך False, לא יוצג פס הגלילה מחדש. הפונקציה מחזירה ערך לא-אפס אם יצאה לפועל בהצלחה, וערך אפס אם לא.

9.5 קביעת מצב הגררה בפס גלילה (SetScrollPos)

פסי גלילה הם פקדים להפעלה ידנית. לפיכך, חייבת התוכנית שלך להזיז את תיבת הגררה על פי הצורך. לשם כך, השתמש בפונקציה ()SetScrollPos, שהגדרתה הכללית:

```
int SetScrollPos(HWND hwnd, int which, int pos, BOOL repaint);
```

כאן, hwnd היא הידית המגדירה את פס הגלילה. בפסי גלילה של חלונות, זוהי ידית החלון; בפסי גלילה שהם פקדים, זוהי הידית של פס הגלילה. הערך של which יקבע איזה פס גלילה יקבל הגדרות לתיבת הגררה שלו. אם אתה עוסק בקביעת מצבה של גררה בפס גלילה אנכי, יש להקצות לפרמטר זה את הערך SB_VERT. אם אתה עוסק בקביעת מצבה של גררה בפס גלילה אופקי, עליך להקצות לפרמטר זה את הערך SB_HORZ. אבל בפקד פס גלילה, חייב ערך זה להיות SB_CTL. הערך של pos יקבע את מיקומה של תיבת הגררה. ערך זה חייב לייצג מיקום המצוי בתוך גבולות פס הגלילה. אם הערך של repaint הוא true, יצויר פס הגלילה מחדש לאחר קביעת מיקום תיבת הגררה. אם הערך הוא false, לא יוצג פס הגלילה מחדש.

9.6 תוכנית ליצירת פסי גלילה (בתוך תיבת דו-שיח)

בסעיף זה תוצג תוכנית פשוטה שתדגים לפניך כיצד ליצור פס גלילה אנכי בתוך תיבת דו-שיח, ולטפל בהודעות ממנו. לתוכנית זו דרוש דו-שיח המוגדר כך:

```
TESTDIALOG DIALOG DISCARDABLE  20, 20, 154, 74
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
```

```
CAPTION "Using a Scroll Bar"
FONT 8, "MS Sans Serif"
BEGIN
    GROUPBOX        "Thumb Position",IDD_GB1,1,1,60,30
    SCROLLBAR       IDD_SB1,140,0,10,70,SBS_VERT | WS_TABSTOP
END
```

לפניך התוכנית **ScrollBar** ליצירת פסי גלילה בשלמותה (בתקליטור Chap09). היא מגיבה להודעות SB_LINEUP, SB_LINEDOWN, SB_PAGEUP, SB_PAGEDOWN, SB_THUMBPOSITION ו- SB_THUMBTRACK בהתאם לתזוזת תיבת הגררה. שים לב שקביעת מצב הגררה הראשוני נעשית כתגובה להודעה WM_INITDIALOG. כמו כן, היא מציגה את המצב הנוכחי של הגררה בתיבת הקבוצה "Thumb Position" (התוכנית משתמשת לצורך זה בפונקציה : ShowPos). מצב זה ישתנה ככל שתזיז את הגררה. בתרשים 9.3 תוכל לראות פלט לדוגמה מהתוכנית. נקודה נוספת: שים לב שמצב הגררה מוצג על ידי משלוח פלט טקסט אל שטח הלקוח של תיבת הדו-שיח בעזרת הפונקציה ()TextOut. על אף שתיבות דו-שיח משרתות מטרה מיוחדת, בסופו של דבר הן חלונות, ולחלונות אלה יש מאפיינים בסיסיים כמו לחלון הראשי .

```
#include <windows.h>
#include <stdio.h>
#include "ScrollBar.h"
#include "resource.h"
#if defined (win32)
   #define IS_WIN32 TRUE
#else
   #define IS_WIN32 FALSE
#endif

#define RANGEMAX 50

char str[255]; /* holds output strings */

int X=0, Y=0; /* current output location */

HINSTANCE hInst;         // current instance
LPCTSTR lpszAppName = "MyApp";
LPCTSTR lpszTitle =   "ScrollBar Inside Dialog Application";
BOOL RegisterWin95(CONST WNDCLASS* lpwc);
void ShowPos(HWND hDlg, int nPos);
```

```
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE
                    hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
 {
   MSG msg;
   HWND hWnd;
   WNDCLASS wc;

   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = (WNDPROC)WndProc;
   wc.cbClsExtra     = 0;
   wc.cbWndExtra     = 0;
   wc.hInstance      = 0;
   wc.hIcon          = LoadIcon(hInstance, lpszAppName);
   wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName   = lpszAppName;
   wc.lpszClassName  = lpszAppName;

   if(!RegisterWin95(&wc))
      return false;
   hInst = hInstance;
   hWnd = CreateWindow (lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                       );
   if(!hWnd)
      return false;
   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
   while(GetMessage(&msg, NULL, 0,0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
   return(msg.wParam);
}
```

```
BOOL RegisterWin95(CONST WNDCLASS* lpwc)
 {
   WNDCLASSEX wcex;

   wcex.style           = lpwc->style;
   wcex.lpfnWndProc     = lpwc->lpfnWndProc;
   wcex.cbClsExtra      = lpwc->cbClsExtra;
   wcex.cbWndExtra      = lpwc->cbWndExtra;
   wcex.hInstance       = lpwc->hInstance;
   wcex.hIcon           = lpwc->hIcon;
   wcex.hCursor         = lpwc->hCursor;
   wcex.hbrBackground   = lpwc->hbrBackground;
   wcex.lpszMenuName    = lpwc->lpszMenuName;
   wcex.lpszClassName   = lpwc->lpszClassName;
   wcex.cbSize          = sizeof(WNDCLASSEX);
   wcex.hIconSm         = LoadIcon(wcex.hInstance, "SMALL");
   return RegisterClassEx(&wcex);
 }

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam)
 {
   switch(uMsg)
    {
      case WM_COMMAND:
         switch(LOWORD(wParam))
          {
            case IDM_TEST :
               DialogBox( hInst, "TestDialog", hWnd,
                          (DLGPROC)TestDlgProc );
               break;
            case IDM_EXIT :
               DestroyWindow(hWnd);
               break;
          }
         break;
      case WM_DESTROY :
         PostQuitMessage(0);
         break;
      default:
         return (DefWindowProc(hWnd, uMsg, wParam, lParam));
    }
   return(0L);
}
```

```
LRESULT CALLBACK TestDlgProc( HWND hDlg, UINT uMsg,
                              WPARAM wParam, LPARAM lParam )
{
   static int pos = 0; /* slider box position */

   switch( uMsg )
   {
   case WM_INITDIALOG:
      SetScrollRange((HWND) GetDlgItem(hDlg,IDD_SB1), SB_CTL, 0,
                     RANGEMAX, 1);
      break;
   case WM_VSCROLL:

      switch(LOWORD(wParam)) {
        case SB_LINEDOWN:
          pos++;
          if(pos>RANGEMAX) pos = RANGEMAX;
          SetScrollPos((HWND) lParam, SB_CTL, pos, 1);
          ShowPos(hDlg, pos);
          break;
        case SB_LINEUP:
          pos--;
          if(pos<0) pos = 0;
          SetScrollPos((HWND) lParam, SB_CTL, pos, 1);
          ShowPos(hDlg, pos);
          break;
        case SB_THUMBPOSITION:
          pos = HIWORD(wParam); /* get current position */
          SetScrollPos((HWND) lParam, SB_CTL, pos, 1);
          ShowPos(hDlg, pos);
          break;
        case SB_THUMBTRACK:
          pos = HIWORD(wParam); /* get current position */
          SetScrollPos((HWND) lParam, SB_CTL, pos, 1);
          ShowPos(hDlg, pos);
          break;
        case SB_PAGEDOWN:
          pos += 5;
          if(pos>RANGEMAX) pos = RANGEMAX;
          SetScrollPos((HWND) lParam, SB_CTL, pos, 1);
          ShowPos(hDlg, pos);
          break;
```

```
        case SB_PAGEUP:
          pos -= 5;
          if(pos<0) pos = 0;
          SetScrollPos((HWND) lParam, SB_CTL, pos, 1);
                        ShowPos(hDlg, pos);
          break;
      }
      break;
      case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
               case IDOK:
                    EndDialog( hDlg, IDOK );
                    break;
               case IDCANCEL :
                    EndDialog( hDlg, IDCANCEL );
                    break;
                                  }
            break;

      default :
            return( FALSE );
   }

   return( TRUE );
}

void ShowPos(HWND hDlg, int nPos) /* Display the position of the
                                     scroll bar. */
{
   HDC hDC;
   char str[80];

   hDC = GetDC(hDlg);

   sprintf(str, "%d", nPos);
   TextOut(hDC, 55, 30, "    ", 4);
   TextOut(hDC, 55, 30, str, strlen(str));

   ReleaseDC(hDlg, hDC);
}
```



תרשים 9.3: דוגמת פלט מהפעלת תוכנית ליצירת פסי גלילה

אצל מתכנתי חלונות מתחילים נוצר משום-מה הרושם, שפסי גלילה הם עניין מסובך. למעשה, פסי גלילה הם אחד מאמצעי הפיקוד הפשוטים ביותר של חלונות.

9.7 הפונקציה ShowScrollBar

למדת שבאפשרות התוכניות להצמיד פסי גלילה לחלונות הן בזמן יצירת החלון והן לאחר שהחלון נוצר. כדי להצמיד או למחוק פס גלילה מחלון שנוצר כבר, משתמשים בפונקציה ShowScrollBar. פונקציה זו מציגה או מוחקת את פס הגלילה שהוגדר. את הפונקציה ShowScrollBar כותבים כמו בהגדרה שלהלן:

```
BOOL ShowScrollBar(
  HWND hWnd,           // handle of window with scroll bar
  int wBar,            // scroll bar flag
  BOLL bShow           // scroll bar visibility flag
);
```

הפרמטר hWnd מזהה פקד של פס גלילה או חלון עם פס גלילה סטנדרטי, על פי הערך של הפרמטר wBar. הפרמטר bShow מגדיר אם Windows מציגה או מוחקת את פס הגלילה. אם bShow שווה ל-True, Windows מציגה את פס הגלילה; אחרת, Windows מוחקת אותו. הפרמטר wBar מגדיר את פס הגלילה ש-Windows תציג או תמחק. פרמטר זה יכול להיות אחד הערכים שמפורטים בטבלה 9.1.

טבלה 9.1: הערכים האפשריים עבור הפרמטר **wBar**.

ערך	פירוש
SB_BOTH	מציג או מוחק את פסי הגלילה הסטנדרטיים האופקיים והאנכיים של החלון.
SB_CTL	מציג או מוחק את פקד פס הגלילה. הפרמטר hWnd חייב להיות הידית של פקד פס הגלילה.
SB_HORZ	מציג או מוחק את פסי הגלילה הסטנדרטיים האופקיים של החלון.
SB_VERT	מציג או מוחק את פסי הגלילה הסטנדרטיים האנכיים של החלון.

✍ **הערה:** אין צורך לקרוא לפונקציה ShowScrollbar כדי להחביא פס גלילה בזמן הטיפול בהודעה של פס גלילה.

לאחר הצגת פסי הגלילה נרצה בדרך כלל לשלוט בצורה שבה המשתמש מפעיל אותם. אחד **הפקדים המקובלים** (Common Controls) שתוסיף ברוב המקרים לפס גלילה הוא הפקד המאפשר הפעלה או אי-הפעלה של אחד החיצים או שניהם בפס הגלילה. כדי לעשות זאת, משתמשים בפונקציה EnableScrollBar. הפונקציה EnableScrollBar מאפשרת הפעלה או אי-הפעלה של אחד החיצים או של שניהם. משתמשים בפונקציה EnableScrollBar בתוכניות כפי שמוצג שלהלן:

```
BOOL EnableScrollBar(
    HWND hWnd,          // handle to window or scroll bar
    UINT wSBflags,      // scroll bar type flag
    UINT uArrows        // scroll bar arrow flag
);
```

הפרמטר hWnd מזהה חלון או פקד של פס גלילה, על פי ערך הפרמטר wSBflags. הפרמטר wSBflags מגדיר את סוג פס הגלילה, והוא יכול להיות אחד הערכים שמפורטים בטבלה 9.1. הפרמטר wArrows מגדיר אם החיצים של פס הגלילה מופעלים או שאינם מופעלים, ומציין איזה חץ מופעל או שאינו מופעל. הפרמטר wArrows יכול להיות אחד הערכים שמפורטים בטבלה 9.2.

טבלה *9.2*: הערכים האפשריים עבור הפרמטר **wArrows**.

ערך	פירוש
ESB_DISABLE_BOTH	גורם לשני חיצי פס הגלילה להיות במצב לא-פעיל.
ESB_DISABLE_DOWN	מעביר את החץ התחתון בפס גלילה אנכי למצב לא-פעיל.
ESB_DISABLE_LEFT	מעביר את החץ השמאלי בפס גלילה אופקי למצב לא-פעיל.
ESB_DISABLE_LTUP	מעביר את החץ השמאלי בפס גלילה אופקי למצב לא-פעיל, או מעביר את החץ העליון בפס גלילה אנכי למצב לא-פעיל.
ESB_DISABLE_RIGHT	מעביר את החץ הימני בפס גלילה אופקי למצב לא-פעיל.
ESB_DISABLE_RTDN	מעביר את החץ הימני בפס גלילה אופקי למצב לא-פעיל, או מעביר את החץ התחתון בפס גלילה אנכי למצב לא-פעיל.
ESB_DISABLE_UP	מעביר את החץ העליון בפס גלילה אנכי למצב לא-פעיל.
ESB_ENABLE_BOTH	גורם לשני החיצים של פס הגלילה להיות במצב פעיל.

שתי פונקציות אלו יחד, נותנות אפשרות לתוכניות לנהל את רוב הפעילות של פס הגלילה הדרוש להן.

9.8 המיקום והטווח של פס הגלילה

ברוב המקרים, היישום משנה את הטווח כדי לשקף את גודל המסמך או התמונה. מיקום ה**גררה** (Thumb) הוא הערך שנמצא בטווח שבו היא נמצאת. לדוגמה, אם הטווח הוא מ-0 עד 100 ומיקום הגררה הוא 50, אזי הגררה נראית באמצע פס הגלילה.

באפשרותך גם לקבוע את **גודל הדף** (Page Size) של פקדי פס הגלילה. גודל הדף מייצג את מספר התוספות בטווח פס הגלילה, שהדף יכול להציג בו-זמנית. לדוגמה, אם הטווח הוא מ-0 עד 100 וגודל הדף נקבע ל-50, פירוש הדבר שהדף יכול להציג חצי מטווח הפקד בכל זמן נתון. יש להשתמש בפונקציות SetScrollInfo ו-GetScrollInfo כדי לקבוע ולקבל את טווח פס הגלילה, מיקום הגררה וגודל הדף, במקום בפונקציות הישנות (SetScrollPos ו- GetScrollPos).

9.9 קבלת הערכים הנוכחיים של פס הגלילה

כפי שלמדת, לפסי גלילה יש ערכי ברירת מחדל מסוימים בעת שמוסיפים אותם לחלונות התוכנית. אך למדת שהתוכניות משנות את הקביעות האלו במהלך הביצוע שלהן. פעמים רבות התוכניות חייבות לבצע פעולה מסוימת המבוססת על קביעות אלו. כדי שתוכנית תוכל לדעת את ההגדרות הנוכחיות של פס הגלילה, עליה להשתמש בפונקציה GetScrollInfo. הפונקציה GetScrollInfo מקבלת את הפרמטרים של פס הגלילה, ובכלל זה את המיקום המינימלי והמקסימלי של הגלילה, את גודל הדף ואת המיקום של **תיבת הגלילה** (Scroll Box, או הגררה). את הפונקציה GetScrollInfo כותבים בתוכנית כפי שמוצג שלהלן:

```
BOOL GetScrollInfo(
  HWND hWnd,          // handle of window with scroll bar
  int fnBar,          // scroll bar flag
  LPSCROLLINFO lpsi   // pointer of structure for scroll
                      // parameters
);
```

כמו שאפשר לראות, הפונקציה GetScrollInfo מקבלת שלושה פרמטרים. הפרמטר lpsi, עליו תלמד יותר בהמשך סעיף זה, מחזיר ערך של SCROLLINFO. זהו מבנה שמוגדר במערכת (A System-Defined Structure), בעל האיברים הבאים:

```
typedef struct tagSCROLLINFO
  {
    UINT cbSize;
    UINT fMask;
    int nMin;
    int nMax;
    UINT nPage;
    int nPos;
    int nTrackPos;
  } SCROLLINFO;
```

שתי הפונקציות GetScrollInfo ו-SetScrollInfo משתמשות במבנה SCROLLINFO. טבלה 9.3 מפרטת את איברי המבנה הזה.

טבלה 9.3: איברי המבנה **SCROLLINFO**.

איבר	תיאור
cbSize	מגדיר את הגודל **בבתים** (Bytes), של המבנה SCROLLINFO.
FMask	מגדיר את הפרמטרים של פס הגלילה, שצריך לקבוע או לקבל. איבר זה יכול להיות שילוב של הערכים שמפורטים בטבלה 9.4.
nMin	מגדיר את מיקום הגלילה המינימלי.
NMax	מגדיר את מיקום הגלילה המקסימלי.
NPage	מגדיר את גודל הדף. פס הגלילה משתמש בערך זה כדי להחליט על הגודל הפרופורציונלי המתאים של **תיבת הגלילה** (Scroll Box).
NPos	מגדיר את המיקום של תיבת הגלילה.
NTrackPos	מגדיר את המיקום המיידי של תיבת הגלילה שהמשתמש גורר כרגע. באפשרות היישום לקבל את הערך הזה בזמן הטיפול בהודעת שינוי המצב SB_THUMBTRACK. היישום אינו יכול לקבוע את מיקום הגלילה המיידי. הפונקציה SetScrollInfo מתעלמת מאיבר זה.

כמו שראית בטבלה 9.3, האיבר fMask יכול לקבל אחד ממספר ערכים מובנים. טבלה 9.4 מציגה את הערכים האפשריים שהפרמטר fMask יכול לקבל.

טבלה 9.4: הערכים המובנים האפשריים שהפרמטר **fMask** יכול לקבל.

ערך	פירוש
SIF_ALL	שילוב של SIF_PAGE, SIF_POS ו-SIF_RANGE.
SIF_DISABLENOSCROLL	משתמשים בערך זה בתוכניות רק כאשר קובעים את הפרמטרים של פס הגלילה. אם הפרמטרים החדשים שקובעים לפס הגלילה גורמים לכך שלא צריך אותו יותר, חייבים להעביר אותו למצב לא-פעיל במקום להסיר אותו.
SIF_PAGE	האיבר nPage מכיל את גודל הדף עבור פס גלילה פרופורציונלי.
SIF_POS	האיבר nPos מכיל את המיקום של תיבת הגלילה.
SIF_RANGE	האיברים nMin ו-nMax מכילים את הערך המינימלי והמקסימלי של טווח הגלילה.

הפרמטר hWnd שבפונקציה GetScrollInfo, מציין פקד פס גלילה או חלון עם **פס גלילה סטנדרטי** (Standard Scroll Bar), על פי ערך הפרמטר fnBar. הפרמטר fnBar מגדיר את סוג פס הגלילה שצריך לקבל עבורו את הפרטרים, והוא יכול לקבל אחד מהערכים שמפורטים בטבלה 9.5.

טבלה 9.5: הערכים האפשריים שהפרמטר **fnBar** יכול לקבל.

ערך	פירוש
SB_CTL	מקבל את הפרמטרים עבור פקד פס גלילה. הפרמטר hWnd חייב להכיל את הידית של פקד פס גלילה.
SB_HORZ	מקבל את הפרמטרים עבור פס גלילה אופקי סטנדרטי של החלון.
SB_VERT	מקבל את הפרמטרים עבור פס גלילה אנכי סטנדרטי של החלון.

לבסוף, הפרמטר lpsi מצביע למבנה מסוג SCROLLINFO שהאיבר fMask שלו, ברגע הכניסה לפונקציה, מגדיר את הפרמטרים של פס הגלילה שצריך לקבל. לפני שחוזרים, הפונקציה מעתיקה את הפרמטרים המוגדרים לאיברים המתאימים של המבנה. הערך של האיבר fMask יכול להיות שילוב של הערכים שמפורטים בטבלה 9.6 (שים לב שהפונקציה GetScrollInfo מקבלת רק ערכים מסוימים של האיבר fMask).

טבלה 9.6: הערכים האפשריים של האיבר **fMask** שבמבנה **SCROLLINFO**.

ערך	פירוש
SIF_PAGE	מעתיק את דף הגלילה לאיבר nPage של המבנה SCROLLINFO שמוצבע על ידי הפרמטר lpsi.
SIF_POS	מעתיק את מקום הגלילה לאיבר nPos של המבנה SCROLLINFO שמוצבע על ידי הפרמטר lpsi.
SIF_RANGE	מעתיק את טווח הגלילה לאיברים nMin ו-nMax של המבנה SCROLLINFO שמוצבע על ידי הפרמטר lpsi.

הפונקציה GetScrollInfo מאפשרת ליישומים להשתמש בערך מיקום גלילה בן 32 סיביות (32-bit Value). למרות שההודעות שמציינות את מיקום פס הגלילה, WM_HSCROLL ו-WM_VSCROLL מאפשרות ערך בן 16 סיביות (16-bit Value) בלבד עבור מיקום הנתונים, הפונקציות SetScrollInfo ו-GetScrollInfo מאפשרות להשתמש בערך של 32 סיביות עבור מיקום הנתונים של פס הגלילה. לכן, היישום יכול לקרוא ל-GetScrollInfo כאשר הוא מטפל בהודעות WM_HSCROLL או WM_VSCROLL, כדי להשיג את מיקום הנתונים של פס הגלילה באמצעות ערך של 32 סיביות.

ההגבלות על השימוש בערך של 32 סיביות עבור מיקום הנתונים של פס הגלילה מתייחסות לגלילת תכולת החלון **בזמן אמת** (Real Time). היישום מיישם גלילת זמן אמת על ידי טיפול בהודעות WM_HSCROLL או WM_VSCROLL, שמכילות את ערך שינוי

המצב SB_THUMBTRACK, וכך הוא עוקב אחר מיקום **תיבת הגלילה** (Thumb) כאשר המשתמש מזיז אותה. לרוע המזל, אין פונקציה שמקבלת ערך בן 32 סיביות של מיקום תיבת הגלילה כאשר המשתמש מזיז אותה. מכיון ש-GetScrollInfo מאפשרת לקבל את המיקום הסטטי בלבד, היישום משיג רק ערך בן 32 סיביות של מיקום הנתונים לפני, או אחרי פעולת הגלילה.

9.10 גלילת תוכן החלון

הפעולה החשובה ביותר שפסי הגלילה מבצעים היא מתן אפשרות למשתמשים לגלול את תוכן החלון. השליטה בגלילת החלון נעשית באמצעות הפונקציה ScrollWindowEx. פונקציה זו גוללת את התכולה שנמצאת בשטח הלקוח של החלון המוגדר. משתמשים בפונקציה ScrollWindowEx בתוכניות כמו שרואים בהגדרה שלהלן:

```
int ScrollWindowEx(
    HWND hWnd,              // handle of window to scroll
    int dx,                 // amount of horizontal scrolling
    int dy,                 // amount of vertical scrolling
    CONST RECT *prcScroll,  // address of structure with
                            // scroll rectangle
    CONST RECT *prcClip,    // address of structure with
                            // clip rectangle
    HRGN hrgnUpdate,        // handle of update region
    LPRECT prcUpdate,       // address of structure for
                            // update rectangle
    UINT flags              // scrolling flags
);
```

הפונקציה ScrollWindowEx מקבלת שמונה פרמטרים, שאת מרביתם אפשר לצפות. טבלה 9.7 מפרטת את הפרמטרים שמקבלת הפונקציה ScrollWindowEx.

טבלה 9.7: הפרמטרים של הפונקציה **ScrollWindowEx**.

פרמטר	תיאור
hWnd	מזהה את החלון שפונקציה ScrollWindowEx אמורה לגלול את שטח הלקוח שלו.
dx	מגדיר את טווח הגלילה האופקית ביחידות התקן. פרמטר זה חייב להיות שלילי כדי לגלול שמאלה.
dy	מגדיר את טווח הגלילה האנכית ביחידות התקן. פרמטר זה חייב להיות שלילי כדי לגלול כלפי מעלה.

פרמטר	תיאור
prcScroll	מצביע למבנה RECT שמגדיר את חלק שטח הלקוח בחלון שפונקציה ScrollWindowEx אמורה לגלול. אם פרמטר זה הוא NULL, הפונקציה ScrollWindowEx גוללת את כל השטח.
prcClip	מצביע למבנה RECT שמכיל את קואורדינטות **מלבן הגזירה** (Clipping Rectangle). הפונקציה ScrollWindowEx משפיעה על חלקי ההתקן שנמצאים במלבן הגזירה. Windows צובעת חלקים שהפונקציה גוללת מחוץ המלבן פנימה; Windows אינה צובעת חלקים שהפונקציה גוללת מתוך המלבן החוצה.
hrgnUpdate	מזהה את האזור שהפונקציה ScrollWindowEx מגדירה כדי להחזיק את האזור שהגלילה הופכת אותו ללא תקף. הפרמטר hrgnUpdate יכול להיות NULL.
prcUpdate	מצביע למבנה מסוג RECT שמקבל את גבולות המלבן שהגלילה הופכת אותו ללא תקף. הפרמטר hrgnUpdate יכול להיות NULL.
flags	מגדיר דגלים ששולטים בגלילה. פרמטר זה מקבל אחד הערכים שמפורטים בטבלה 9.8.

כמו שציינו בטבלה 9.7, הפרמטר flags יכול לקבל אחד ממספר ערכים מובנים שבמערכת. טבלה 9.8 מפרטת את הערכים האפשריים של הפרמטר flags.

אם הפונקציה מצליחה, הערך המוחזר הוא SIMPLEREGION (אזור מלבני שאינו תקף), COMPLEXREGION (אזור שאינו מלבני ואינו תקף; המלבנים חופפים), או NULLREGION (אין אזור תקף); ואם הפונקציה נכשלת, היא מחזירה את הערך ERROR.

טבלה 9.8: הערכים האפשריים עבור הפרמטר **flags**.

ערך	פירוש
SW_ERASE	מוחק את האזור החדש הלא-תקף, על ידי שליחת ההודעה WM_ERASEBKGND לחלון כאשר כוללים את הדגל SW_ERASE עם הדגל WS_INVALIDATE.
SW_INVALIDATE	גורם לאי תקפות האזור שמזוהה על ידי הפרמטר hrgnUpdate.
SW_SCROLLCHILDREN	גולל את כל חלונות הבנים שנחתכים (Intersect, חיתוך) עם המלבן שמצביע עליו הפרמטר prcScroll. Windows גוללת את חלונות הבנים כמספר הפיקסלים שמצוין בפרמטרים dx ו-dy, ושולחת הודעת WM_MOVE לכל החלונות הבנים שנחתכים עם המלבן prcScroll, אפילו אם הם אינם זזים.

אם בקריאה לפונקציה מוגדרים הדגלים SW_INVALIDATE ו-SW_ERASE, ScrollWindowEx אינה מסמנת את האזור שהיא גוללת כלא תקף. אם לא נקבע אף לא אחד משני דגלים אלה, ScrollWindowEx מסמנת את האזור שהיא גוללת כלא תקף. Windows אינה מעדכנת את האזור עד שהיישום קורא לפונקציה UpdateWindow, או שהוא קורא לפונקציה RedrawWindow (מגדיר את RDW_UPDATENOW או את הדגל RDW_ERASENOW), או מקבל ההודעה WM_PAINT מתור ההודעות של היישום.

אם יש לחלון את הסגנון WS_CLIPCHILDREN, האזורים המוחזרים שמוגדרים על ידי hrgnUpdate ו-prcUpdate מייצגים את האזור הכולל שעל Windows לעדכן, ובכלל זה אזורים שבחלונות הבן. אם בקריאה לפונקציה ScrollWindowEx מוגדר הדגל SW_SCROLLCHILDREN, אזי Windows אינה מעדכנת את המסך כראוי אם המשתמש גולל רק חלק של חלון בן. Windows אינה מוחקת את החלק הנגלל של חלון הבן שנמצא מחוץ למלבן המקור, ואינה צובעת מחדש את חלון הבן ביעד החדש שלו. כדי להזיז חלונות הבן שאינם נמצאים כולם במלבן שמוגדר על ידי prcScroll, השתמש בפונקציה DeferWindowPos. Windows ממקמת את הסמן מחדש אם נקבע הדגל SW_SCROLLCHILDREN ומלבן סמן הטקסט חותך (Intersect) את מלבן הגלילה.

Windows מציינת את כל קואורדינטות הקלט והפלט (עבור prcScroll, prcClip, prcUpdate ו-hrgnUpdate) כקואורדינטות לוגיות, ללא קשר אם יש לחלון את סגנון המחלקה CS_OWNDC או CS_CLASSDC. השתמש בפונקציות LPtoDP ו-DPtoLP כדי לתרגם מקואורדינטות לוגיות ולקואורדינטות לוגיות, אם יש צורך בכך.

כדי להבין יותר טוב את פעולת הפונקציה ScrollWindowEx, התבונן בתוכנית **Scroll_Window** שנמצאת בתקליטור המצורף לספר זה (Chap09). התוכנית **Scroll_Window** מאפשרת למשתמש להוסיף מדי פעם שורה לחלון התוכנית. כשמספר הקווים עובר את השטח הקיים במרחב החלון, פס הגלילה הופך לפעיל. כאשר המשתמש לוחץ בעכבר על החץ העליון או על החץ התחתון, הגררה זזה שורה אחת בכיוון המתאים.

9.11 ההודעה WM_SIZE

בסעיף הקודם למדת על התוכנית **Scroll_Window**, שאיפשרה למשתמשים לבחור בפס הגלילה כדי לנוע למעלה ולמטה בחלון. התוכנית מפעילה את פס הגלילה כאשר תכולת החלון ארוכה מדי מכדי להציג אותה בגודל החלון הנוכחי. אם ניסית את התוכנית **Scroll_Window** והגדלת את החלון, ודאי ראית שפס הגלילה הפך להיות שוב לא-פעיל, לאחר שהגדלת את החלון במידה מספיקה להצגת כל הטקסט. התוכנית **Scroll_Window** לוכדת את הודעת המערכת WM_SIZE ומעדכנת את פס הגלילה, לאחר ששינית את גודל החלון:

```
// Every time the window is sized, re-calculate the number of
// lines the client area can display and set the scroll bar
// accordingly.
```

```
case WM_SIZE :
    {
       RECT rect;
       GetClientRect( hWnd, &rect );

       nDspLines = rect.bottom / 20;
       if ( nDspLines < nNumItems )
        {
          SCROLLINFO si;

          si.cbSize = sizeof( SCROLLINFO );
          si.fMask  = SIF_POS | SIF_RANGE | SIF_PAGE;
          si.nMin   = 0;
          si.nMax   = nNumItems-1;
          si.nPage  = nDspLines;
          si.nPos   = nCurPos;
          EnableScrollBar(hWnd, SB_VERT, ESB_ENABLE_BOTH);
          SetScrollInfo( hWnd, SB_VERT, &si, TRUE );
        }
       else
        EnableScrollBar(hWnd, SB_VERT, ESB_DISABLE_BOTH);
    }
    break;
```

במקרה של התוכנית **Scroll_Window**, משפט case של WM_SIZE בודק את מספר השורות שיכול החלון להציג כנגד מספר השורות שמוצגים כרגע. אם מספר השורות האפשרי גדול ממספר השורות המוצגות, התוכנית מעבירה את פס הגלילה למצב לא-פעיל; אחרת, התוכנית משנה את גודל הגררה (אם יש צורך בכך) ומציגה את פס הגלילה החדש שהותאם לגודל החלון.

משתמשים בהודעה WM_SIZE כדי לבדוק את גודל החלון הנוכחי ולהחליף פריטים כלשהם שנמצאים בחלון שהמקום שלו תלוי בגודל החלון. לדוגמה, אם יש לך **תיבת עריכה** (Edit Box) בחלון שמציג כותרת והמשתמש משנה את גודל החלון, תרצה בוודאי לשנות את גודל תיבת העריכה בהתאמה עם החלון. הפעולה שיבצעו התוכניות כאשר הן מקבלות את ההודעה WM_SIZE תהיה שונה, בהתבסס על סוג שינוי הגודל שעשה המשתמש. ההודעה WM_SIZE מעבירה קבוע שמייצג את סוג שינוי הגודל בפרמטר wParam. טבלה 9.9 מפרטת את הערכים האפשריים של הפרמטר wParam.

טבלה 9.9: הערכים האפשריים של הפרמטר **wParam** בהודעה **WM_SIZE**.

ערך	פירוש
SIZE_MAXHIDE	Windows שולחת את הודעה לכל **החלונות הנשלפים** (Pop-Up Window) כשהמשתמש מגדיל למקסימום חלון אחר כלשהו.
SIZE_MAXIMIZED	המשתמש הגדיל את החלון לגודל מקסימום.
SIZE_MAXSHOW	Windows שולחת את הודעה לכל החלונות הנשלפים כאשר המשתמש משחזר חלון אחר כלשהו לגודל שהיה לו קודם.
SIZE_MINIMIZED	המשתמש הקטין את החלון לגודל מינימום.
SIZE_RESTORED	המשתמש שינה את גודל החלון, אבל אף לא אחד מהערכים SIZE_MAXIMIZED או SIZE_MINIMIZED ישימים.

בנוסף, ההודעה WM_SIZE מעבירה מידע על החלון החדש בפרמטר lParam. מילת הסדר-הנמוך של lParam מגדירה את הרוחב החדש של שטח הלקוח, ומילת הסדר-הגבוה של lParam מגדירה את הגובה החדש של שטח הלקוח.

9.12 ההודעה WM_PAINT

למדת שהיישום מקבל את ההודעה WM_SIZE בכל פעם שהמשתמש משנה את גודל החלון. Windows גם תשלח ליישום את ההודעה WM_PAINT אחרי ההודעה WM_SIZE בכל פעם שהמשתמש משנה את גודל החלון. למעשה, היישום מקבל את ההודעה WM_PAINT כאשר Windows, או יישום אחר, מבקש לצבוע חלק מחלון היישום שלך. Windows שולחת את ההודעה WM_PAINT כשהתוכנית קוראת לפונקציה UpdateWindow או ל-RedrawWindow; או שהיא שולחת את ההודעה לפונקציה DispatchMessage כאשר היישום משתמש בפונקציה GetMessage או בפונקציה PeekMessage כדי לקבל את ההודעה WM_PAINT.

הפרמטר wParam של ההודעה WM_PAINT מכיל את הערך hdc. הערך hdc הוא ידית ל**הקשר התקן** (Device Context). פרמטר זה מזהה את הקשר ההתקן שבו Windows צובעת. אם הפרמטר wParam הוא NULL, היישום צריך להשתמש בהקשר ההתקן של ברירת המחדל (במקום ליצור **הקשר התקן פרטי**, Private Device Context). מספר פקדים משותפים משתמשים בפרמטר wParam כדי שיהיה אפשר לצייר בהקשר התקן אחר מזה של ברירת המחדל. חלונות אחרים יכולים להתעלם בצורה בטוחה מ-wParam.

הפונקציה DefWindowProc הופכת את **אזור העדכון** (Update Region) **לתקף** (Validate). יכול להיות שהפונקציה תשלח גם הודעת WM_NCPAINT לפונקציית החלון, אם Windows חייבת לצבוע את **חלון המסגרת** (Window Frame), ובנוסף היא שולחת את ההודעה WM_ERASEBKGND אם Windows חייבת למחוק את הרקע של החלון.

המערכת שולחת את ההודעה WM_PAINT כאשר אין הודעות אחרות בתור ההודעות של היישום. DispatchMessage מחליטה לאן לשלוח את ההודעה. GetMessage מחליטה איזו הודעה לשלוח. GetMessage מחזירה את ההודעה WM_PAINT כאשר אין הודעות אחרות בתור ההודעות של היישום, ו-DispatchMessage שולחת את ההודעה לפונקציית החלון המתאימה.

יכול להיות שהחלון יקבל הודעות WM_PAINT פנימיות כתוצאה מהקריאה לפונקציה RedrawWindow עם הדגל RDW_INTERNALPAINT. במקרה כזה, ייתכן שלחלון אין אזור עדכון. היישום צריך לקרוא לפונקציה GetUpdateRect כדי להחליט אם יש לחלון אזור עדכון. אם GetUpdateRect מחזירה אפס, היישום אינו צריך לקרוא לפונקציות BeginPaint ו-EndPaint. אם היישום אינו קורא לפונקציה GetUpdateRect, אזור העדכון לא יעודכן כראוי, או שלא יעודכן כלל.היישום חייב לחפש במבנה הנתונים הפנימיים שלו כל הודעה WM_PAINT כדי לבדוק אם יש צורך בצביעה פנימית כלשהי. עליו לעשות זאת משתי סיבות: ייתכן שאזור העדכון שאינו NULL, או קריאה לפונקציה RedrawWindow עם הדגל RDW_INTERNALPAINT היא שגרמה להודעה WM_PAINT.

Windows שולחת הודעה WM_PAINT פנימית רק פעם אחת. אחרי ש-GetMessage מחזירה הודעת WM_PAINT פנימית, או לאחר ש-UpdateWindow שולחת PeekMessage לחלון, Windows אינה משגרת הודעות WM_PAINT נוספות. היא ממשיכה לשלוח הודעות רק לאחר שהתוכנית מודיעה **שהחלון לא תקף** (Invlidates The Window), או עד שהתוכנית קוראת פעם נוספת ל-RedrawWindow עם הדגל RDW_INTERNALPAINT.

בחלק מהפקדים המשותפים, עיבוד הודעת ברירת המחדל WM_PAINT כולל את בדיקת הפרמטר wParam. אם wParam אינו NULL, הפקדים מניחים שהערך הוא ידית של הקשר התקן והם ישתמשו בהקשר ההתקן הזה לצביעה. כדי להבין טוב יותר את הטיפול של התוכנית בהודעה WM_PAINT, התבונן בתוכנית **Scroll_Window** שהוצגה בסעיף 9.11.

9.13 שינוי מצב פסי הגלילה: פעיל ולא-פעיל

התוכניות יכולות לנהל בקרה ושליטה מוחלטת על פסי הגלילה. אחת הפעילויות הנפוצות המבוצעות על ידי התוכניות בעבודה עם פסי גלילה היא להביאם ל**מצב פעיל** (Enable) או ל**מצב לא-פעיל** (Disable). כפי שראית בסעיף 9.11, לדוגמה, התוכנית יכולה להפוך את פס הגלילה ל"לא-פעיל", ולמעשה לבטל אותו, כאשר המשתמש משנה את גודל החלון עד שהוא יכול להציג את כל מה שמוכל בו. כדי להביא את פסי הגלילה למצב פעיל ולמצב לא-פעיל, משתמשים בפונקציה EnableScrollBar. הפונקציה EnableScrollBar גורמת לחץ אחד או יותר, של פס הגלילה, להיות במצב פעיל או במצב לא-פעיל. משתמשים בפונקציה EnableScrollBar בתוכניות כמו שרואים בהגדרה הבאה.

```
BOOL EnableScrollBar(
    HWND hWnd,          // handle to window or scroll bar
    UINT wSBflags,      // scroll bar type flag
    UINT wArrows        // scroll bar arrow flag
);
```

כפי שאפשר לראות, הפונקציה EnableScrollBar מקבלת שלושה פרמטרים. הפרמטר hWnd מזהה חלון או פקד פס גלילה, על פי ערך הפרמטר wSBflags. הפרמטר wSBflags מגדיר את סוג פס הגלילה. הערכים של פרמטר זה יכולים להיות אחד מאלה שמוצגים בטבלה 9.10.

טבלה 9.10: הערכים האפשריים של הפרמטר **wSBflags**.

ערך	פירוש
SB_BOTH	מאפשר הפעלה ואי-הפעלה (הבאה למצב פעיל או למצב לא-פעיל) של החיצים שבפסי הגלילה האופקיים והאנכיים שמוגדרים בחלון מסוים. הפרמטר hWnd חייב להיות ידית החלון.
SB_CTL	מזהה את פס הגלילה כפקד פס גלילה. הפרמטר hWnd חייב להיות הידית של פקד פס הגלילה.
SB_HORZ	מאפשר הפעלה ואי-הפעלה של החיצים בפס הגלילה האופקי שבחלון. הפרמטר hWnd חייב להיות ידית החלון.
SB_VERT	מאפשר הפעלה ואי-הפעלה של החיצים בפס הגלילה האנכי שבחלון. הפרמטר hWnd חייב להיות ידית החלון.

לבסוף, הפרמטר wArrows מגדיר אם חיצי פס הגלילה הם במצב פעיל או במצב לא-פעיל, ומציין איזה חיצים צריכה הפונקציה EnableScrollBar להעביר למצב פעיל או למצב לא-פעיל. הפרמטר wArrows יכול לקבל את אחד הערכים שמפורטים בטבלה 9.11. שים לב שכל הפרמטרים מתייחסים לביטול הפעלה של אחד החיצים, או שניהם.

טבלה 9.11: הערכים האפשריים של הפרמטר **wArrows**.

ערך	פירוש
EBS_DISABLE_BOTH	מבטל הפעלת שני החיצים של פס הגלילה.
EBS_DISABLE_DOWN	מבטל הפעלת החץ התחתון בפס גלילה אנכי.
EBS_DISABLE_LEFT	מבטל הפעלת החץ השמאלי בפס גלילה אופקי.
EBS_DISABLE_LTUP	מבטל הפעלת החץ השמאלי בפס גלילה אופקי, או מבטל הפעלת החץ העליון בפס גלילה אנכי.
EBS_DISABLE_RIGHT	מבטל הפעלת החץ הימני בפס גלילה אופקי.

ערך	פירוש
EBS_DISABLE_RTDN	מבטל הפעלת החץ הימני בפס גלילה אופקי, או מבטל הפעלת החץ התחתון בפס גלילה אנכי.
EBS_DISABLE_UP	מבטל הפעלת החץ העליון בפס גלילה אנכי.
EBS_DISABLE_BOTH	מבטל הפעלת שני החיצים בפס גלילה.

אם הפונקציה מצליחה להביא את החיצים שמוגדרים על ידי הפרמטר wArrows למצב פעיל או למצב לא-פעיל, הערך המוחזר ממנה אינו אפס. אם החיצים נמצאים כבר במצב כנדרש, או שארעה שגיאה, הערך המוחזר הוא אפס.

בתקליטור שמצורף לספר זה תמצא את התוכנית **Enable_Disable**, שגורמת לפס הגלילה האנכי שיהיה במצב פעיל או במצב לא-פעיל כתלות בגודל החלון ותכולתו. שים לב כיצד התוכנית בודקת בעזרת הפונקציה WndProc את גודל החלון ותוכנו בכל פעם שהחלון מקבל את הפקודה WM_SIZE. תוכל לראות שפסי הגלילה עוברים למצב פעיל או למצב לא-פעיל, בהתאמה, לאחר שהחלון מקבל את ההודעה WM_SIZE והתוכנית מטפלת בפקודה זו.

✍ **הערה:** התוכנית לא מגיבה על לחיצות העכבר על החיצים.

9.14 הפונקציה ScrollDC

בסעיפים הקודמים למדת על מספר פונקציות של פסי הגלילה ועל ההודעות שהן יוצרות. בשלב זה חשוב להבין איך יכולות התוכניות לגלול חלונות שמציגים גרפיקה או פריטים שאינם שורות טקסט. ככלל, כאשר מציירים בחלון פריטים שאינם טקסט, משתמשים בהקשר התקן כדי לעשות זאת. אפשר גם להשתמש בהקשר התקן כאשר מציירים טקסט בחלון, למרות שמשתמשים בדרך כלל בגישה בלתי ישירה אל התוכן. כאשר גוללים הקשר התקן בחלון, משתמשים בדרך שונה לגלילת הקשר ההתקן או חלק מהקשר ההתקן. הפונקציה ScrollDC גוללת מלבן של סיביות בהקשר ההתקן בכיוון אופקי ואנכי. את הפונקציה ScrollDC כותבים בתוכניות כמו בדוגמה שלהלן:

```
BOOL ScrollDC(
  HDC hDC,               // handle of device context
  int dx,                // horizontal scroll units
  int dy,                // vertical scroll units
  CONST RECT *lprcScroll, // address of structure for
                         // scrolling rectangle
  CONST RECT *lprcClip,  // address of structure for
                         // clipping rectangle
  HRGN hrgnUpdate,       // handle of scrolling region
  LPRECT lprcUpdate,     // address of structure for
                         // update rectangle
);
```

הפרמטרים שמקבלת הפונקציה ScrollDC מפורטים בטבלה 9.12.

טבלה 9.12: הפרמטרים שמקבלת הפונקציה **ScrollDC**.

פרמטר	תיאור
hDC	מציין את הקשר ההתקן שמכיל את הסיביות שהפונקציה ScrollDC צריכה לגלול.
dx	מגדיר את הגודל, ביחידות התקן, של גלילה אופקית. פרמטר זה חייב להיות ערך שלילי כדי לגלול שמאלה.
dy	מגדיר את הגודל, ביחידות התקן, של גלילה אנכית. פרמטר זה חייב להיות ערך שלילי כדי לגלול כלפי מעלה.
lprcScroll	מצביע למבנה מסוג RECT שמכיל את הקואורדינטות של המלבן הנגלל (Scrolling Rectangle).
lprcClip	מצביע למבנה מסוג RECT שמכיל את הקואורדינטות של מלבן הגזירה (Clipping Rectangle). הפונקציה ScrollDC משפיעה על חלקי ההתקן שמוגדרים במלבן הגזירה. Windows צובעת חלקים שהפונקציה גוללת מחוץ למלבן ופנימה; Windows אינה צובעת חלקים שהפונקציה גוללת מפנים המלבן והחוצה.
hrgnUpdate	מציין את האזור שאינו מכוסה על ידי תהליך הגלילה. ScrollDC מגדירה את האזור הזה, והוא אינו בהכרח מלבני.
lprcUpdate	מצביע למבנה מסוג RECT שמקבל את הקואורדינטות של המלבן שתוחם (Bounding Rectangle) את אזור העדכון (Update Region) הנגלל. זהו השטח המלבני הגדול ביותר שיש צורך בצביעתו. כאשר הפונקציה חוזרת, הערכים שבמבנה נמצאים בקואורדינטות לוגיות, ללא תלות במצב המיפוי של הקשר ההתקן שמוגדר. הדבר מאפשר ליישומים להשתמש באזור העדכון בעת קריאה לפונקציה InvalidateRgn, אם יש צורך בכך.

אם הפרמטר lprcUpdate הוא NULL, Windows אינה מחשבת את מלבן העדכון. אם שני הפרמטרים hrgnUpdate ו-lprcUpdate שווים ל-NULL, Windows אינה מחשבת את אזור העדכון. אם הפרמטר hrgnUpdate אינו NULL, Windows ממשיכה כאילו יש לה ידיעת תקפה לאזור שתהליך הגלילה (שהוגדר על ידי ScrollDC) אינו מכסה. כדי להבין טוב יותר את הטיפול שמבצעת הפונקציה ScrollDC, התבונן בתוכנית **Scroll_DC** שנמצאת בתקליטור המצורף לספר זה. התוכנית מציירת סדרת קווים בחלון; ואז, כאשר מנסים לגלול את החלון עם פס הגלילה, התוכנית גוללת רק חלק חלק מהחלון.

למרות שיכול להיות שקוד התוכנית **Scroll_DC** אינו ברור כל כך, הפונקציה מבצעת פעולה פשוטה למדי: גלילת חלק מהחלון. התוכנית **Scroll_DC** יוצרת הקשר התקן ואחר כך קובעת שהאזור ש-ScrollDC מעבירה יהיה קטן ב-20 יחידות משטח הלקוח של החלון. אחר כך התוכנית קוראת לפונקציה ScrollDC כדי לגלול את המלבן ימינה.

פרק 10

הטיפול בטקסט

בפרק זה נעסוק **בשטח הלקוח** (Client Area) של החלון, ונבדוק היבטים שונים של ניהול טקסט במסגרת חלונות 9x. גם נפתח כמה טכניקות חשובות המפשטות את הצביעה מחדש של חלון לאחר ששוכתב. לסיום, נדון בפרק זה כיצד להשתמש ביישום בגופני טקסט נוספים.

כמו ברוב ההיבטים האחרים של סביבת חלונות 9x, נתונה בידי המתכנת שליטה כמעט בלתי מוגבלת על הדרך שבה יוצג הטקסט וינוהל במסגרת שטח הלקוח של כל חלון נתון. לפיכך, אין באפשרותנו לעסוק בפרק זה בכל ההיבטים של הטיפול בטקסט באמצעות תוכנת חלונות. עם זאת, תוכל בקלות להמשיך ולחקור היבטים נוספים של הטיפול בטקסט לאחר שתבין את היסודות שנציג בפרק זה.

נפתח את הפרק בדיון במערכת הקואורדינטות של החלון, וכיצד הטקסט ממופה עליה. לאחר מכן נעסוק בפונקציות API לטקסט ולמסך. פונקציות אלו יסייעו לך בבקרה ובניהול של פלט הטקסט אל שטח הלקוח של החלונות.

10.1 הקואורדינטות של חלון

הפונקציה ()TextOut היא הפונקציה החלונאית לפלט בצורת טקסט. היא מציגה מחרוזות בקואורדינטות שהוגדרו עבורה, שתמיד מתייחסות לחלון עצמו. לכן, מיקום החלון על המסך אינו משפיע כלל על הקואורדינטות המועברות לפונקציה ()TextOut. לפי ברירת המחדל, הפינה השמאלית-העליונה של שטח הלקוח בחלון נמצאת בנקודה 0, 0. הערך X גדל ככל שנעים ימינה, והערך Y גדל ככל שנעים כלפי מטה.

עד כה השתמשת בקואורדינטות החלונות עבור הפונקציה ()TextOut וכדי למקם מרכיבים שונים בתוך תיבות דו-שיח, מבלי להזכיר באופן מיוחד למה מתייחסות הקואורדיאטות האלו. בנקודה זו רצוי להבהיר כמה פרטים. ראשית, הקואורדינטות המוגדרות עבור ()TextOut הן **קואורדינטות לוגיות** (Logical Coordinates). כלומר, היחידות המשמשות את הפונקציה (ופונקציות נוספות לתצוגה בחלון, לרבות הפונקציות הגרפיות בהן נדון בפרק הבא) הן **יחידות לוגיות** (Logical Units). חלונות ממפה את היחידות הלוגיות והופכת אותן לפיקסלים בעת התצוגה בפועל של הפלט.

עד כה לא היה צורך לעסוק בהבדל הזה כלל, מכיון שלפי ברירת המחדל יחידות לוגיות זהות לפיקסלים. ובכל זאת, חשוב להבין שניתן לבחור במצבי מיפוי שונים שבהם ברירת המחדל הזו, הנוחה בדרך כלל, לא תתאים.

10.2 הגדרת צבע הטקסט והרקע

לפי ברירת המחדל, פלט טקסט לחלון באמצעות הפונקציה ()TextOut יופיע בשחור על גבי הצבע הנוכחי של הרקע. אך תוכל לקבוע את צבע הטקסט ואת צבע הרקע באמצעות הפונקציות ()SetTextColor ו-()SetBkColor, שהגדרותיהן מובאות להלן:

```
COLORREF SetTextColor(HDC hdc, COLORREF color);
COLORREF SetBkColor(HDC hdc, COLORREF color);
```

הפונקציה ()SetTextColor קובעת את צבע הטקסט הנוכחי בהתקן הקשור עם hdc לפי הערך שמכיל הפרמטר color (או צבע קרוב לערך זה, שההתקן מסוגל להציג). הפונקציה ()SetBkColor קובעת את צבע הרקע לטקסט, לפי הערך השמור בפרמטר color (או גוון קרוב לו). עבור שתי הפונקציות, מוחזר הערך של הצבע הקודם. במקרה של שגיאה, מוחזר הערך CLR_INVALID.

הצבע מוגדר כערך מהסוג COLORREF, שהוא מספר שלם בן 32 סיביות. חלונות מאפשרת להגדיר צבעים בשלוש דרכים. הראשונה, והנפוצה ביותר, לפי ערכי RGB (אדום, ירוק, כחול). בשיטה זו, נעשה שילוב בין העוצמות היחסיות של שלושת הצבעים, והצירוף הוא שמחולל את הצבע הרצוי. שיטה שנייה, להגדיר צבע כערך אינדקס בטבלת צבעים לוגית. שיטה שלישית, ערך RGB המתייחס לטבלת צבעים. בפרק זה, תעסוק בשיטה הראשונה בלבד.

אל הפונקציה ()SetTextColor או הפונקציה ()SetBkColor מועבר מספר שלם ארוך, המכיל צבע RGB:

בית (Byte)	**הצבע**
בית 0 (בית מסדר-נמוך)	אדום
בית 1	ירוק
בית 2	כחול
בית 3 (בית מסדר-גבוה)	חייבת להכיל 0

כל צבע בערך RGB מצוי בטווח שבין 0 ל-255, כאשר 0 מייצג את העוצמה הנמוכה ביותר, ו-255 מייצג את העוצמה הגבוהה ביותר. לדוגמה, המספר השלם הארוך שלפניך מפיק צבע מגנטה בוהק:

00	255	00	255

אף שמותר לך בהחלט לעצב ידנית ערך COLORREF, חלונות מגדירה את המאקרו ()RGB המבצע עבורך את העבודה. להלן מתכונתו הכללית:

```
COLORREF RGB(int red, int green, int blue);
```

בדוגמה זו, red, green ו-blue חייבים להכיל ערכים בטווח שבין 0 ל-255. לכן, כדי ליצור צבע מגנטה בוהק, השתמש ב-(255, 0, 255)RGB וכדי ליצור לבן, השתמש ב-(255, 255, 255)RGB. כדי ליצור שחור, השתמש ב- (0, 0, 0)RGB. ליצירת צבעים אחרים, נסה צירופים שונים של שלושת צבעי היסוד בעוצמות משתנות. לדוגמה, המאקרו (100, 100, 0)RGB יוצר צבע טורקיז בהיר. תוכל לערוך ניסיונות כדי לקבוע צבעים המתאימים ליישום שלך.

10.3 כיצד לקבוע את צבע הרקע לתצוגה

באמצעות הפונקציה ()SetBkMode באפשרותך לשלוט בהשפעה שתהיה לטקסט שבמסך על צבע הרקע. לפניך הגדרתה הכללית:

```
int SetBkMode(HDC hdc, int mode);
```

הפונקציה קובעת מה יקרה לצבע הרקע הנוכחי כאשר טקסט (וסוגים אחרים של פלט) יוצג על המסך. הפרמטר hdc מכיל את הידית להֶקשר ההֶתקן המושפע. הפרמטר mode מכיל את הערך הקובע את מצב הרקע, וערך זה חייב להיות אחד משתי פקודות המאקרו: OPAQUE או TRANSPARENT. במקרה של שגיאה, הפונקציה מחזירה את ההגדרה הקודמת, או ערך 0.

אם הערך ב-mode הוא OPAQUE, אזי בכל מקרה של פלט טקסט, הרקע ישתנה לצבע הרקע הנוכחי. אם הפרמטר mode מכיל את הערך TRANSPARENT, אזי צבע הרקע לא ישתנה. במקרה זה, אין שום השפעה לפונקציה ()SetBkColor. לפי ברירת המחדל, מצב הרקע הוא OPAQUE.

10.4 כיצד לקבל את נתוני הטקסט

לתווים אין גודל אחיד. כלומר, בחלונות רוב גופני הטקסט הם יחסיים. לפיכך התו i אינו תופס אותו שטח כמו התו w. כמו כן, גובהם של תווים ואורכם של הקווים היורדים (החלקים ה"תלויים" באותיות כמו p ו-g) אינו זהה בכל הגופנים. נוסף לכך, ניתן לשנות את גודל המרווח בין הקווים האופקיים בכל תו. העובדה שמאפיינים אלה (ונוספים) ניתנים לשינוי אינה חשובה במיוחד, אלא שחלונות דורשת ממך, כמתכנת, לנהל ידנית כמעט את כל הפלט שמוצג כטקסט.

חלונות מספקת רק תמיכה מזערית לפלט טקסט אל שטח הלקוח בחלון. פונקציית הפלט העיקרית היא ()TextOut. הפונקציה מסוגלת להציג רק מחרוזות טקסט שתחילתה בנקודה מוגדרת. אין היא מסוגלת לפרמט פלט, או לבצע באופן אוטומטי פעולה כמו החזרת שורה או מעבר שורה. ניהול הפלט המיועד לשטח הלקוח של כל חלון נתון לחלוטין בידך.

לאור העובדה שלכל גופן גודל משלו (והגופנים עשויים להשתנות תוך כדי ההרצה של התוכנית), חייבת להיות דרך לקבוע את גודל הגופן הנוכחי ומספר תכונות נוספות שלו. לדוגמה, מהעובדה שאפשר לכתוב טקסט בשורות עוקבות משתמע, שיש דרך לברר את גובה הגופן ואת מספר הפיקסלים המפרידים בין שורה לשורה. פונקציית API המשיגה את המידע על הגופן הנוכחי נקראת ()GetTextMetrics, הגדרתה היא:

```
BOOL GetTextMetrics(HDC hdc, LPTEXTMETRIC lpTAttrib);
```

במקרה זה, הפרמטר hdc היא ידית להתקן הפלט, והערך שהוא מכיל מושג בדרך כלל באמצעות הפונקציה ()GetDC, או הפונקציה ()BeginPaint. lpTAttrib הוא מצביע אל מבנה מסוג TEXTMETRIC, שיכיל עם חזרתו את נתוני מידות הגופן הנוכחי. לפניך הגדרת מבנה TEXTMETRIC:

```
typedef struct tagTEXTMETRIC
{
  LONG tmHeight; /* total height of font */
  LONG tmAscent; /* height above base line */
  LONG tmDescent; /* length of descenders */
  LONG tmInternalLeading; /* space above characters */
  LONG tmExternalLeading; /* space between rows */
  LONG tmAveCharWidth; /* average width */
  LONG tmMaxCharWidth; /* maximum width */
  LONG tmWeight; /* weight */
  LONG tmOverhang; /* extra width added to special fonts */
  LONG tmDigitizedAspectX; /* horizontal aspect */
  LtmDigitizedAspectY; /* vertical aspect */
  BYTE tmFirstChar; /* first character in font */
  BYTE tmLastChar; /* last character in font */
  BYTE tmDefaultChar; /* default character */
  BYTE tmBreakChar; /* character used to break words */
  BYTE tmItalic; /* non-zero if italic */
  BYTE tmUnderlined; /* non-zero if underlined */
  BYTE tmStruckOut; /* non-zero if struckout */
  BYTE tmPitchAndFamily; /* pitch and family of font */
  BYTE tmCharSet; /* character set identifier */
} TEXTMERIC;
```

רוב הערכים שהפונקציה משיגה לא ישמשו אותך בפרק זה, אך שניים מהם חשובים מאוד, מכיון שהם משמשים לחישוב המרווח האנכי בין שורות הטקסט. ערך זה הכרחי אם אתה מתכוון להוציא פלט בן יותר משורה אחת לחלון. יישומים המבוססים על עמדות דמויות מכונת כתיבה (Consoles), מאפשרים רק גופן אחד בעל גודל קבוע, לעומת זאת, כאן ייתכנו מספר סוגי גופנים, שונים זה מזה במידותיהם ובצורתם.

כל גופן מגדיר את גובה התווים שלו ואת המרווח הנדרש בין שורות טקסט. כלומר, אי אפשר לדעת מראש את ערך הקואורדינטה האנכית (Y) של שורת הטקסט הבאה. על מנת לקבוע היכן היא תתחיל, עליך להפעיל את הפונקציה ()GetTextMetrics, כדי להשיג שני ערכים: גובה התווים והמרווח בין שורות טקסט. ערכים אלה שמורים בשדות tmHeight ו-tmExternalLeading, בהתאמה. חיבור שני הערכים נותן לך את מספר היחידות האנכיות בין שורה לשורה.

הערה: זכור: הערך שמכיל השדה tmExternalLeading, הוא למעשה, מספר היחידות האנכיות שיש להשאירן ריקות בין שורת טקסט אחת לבאה אחריה. ערך זה אינו זהה לגובה הגופן. אם כך, דרושים לך שני הערכים כדי שתוכל לחשב היכן תתחיל שורת הטקסט הבאה. להלן תראה יישום של עקרון זה.

קיימת גירסה משופרת של TEXTMETRIC, ששמה NEWTEXTMETRIC. הגירסה החדשה זהה לישנה, פרט לכך שנוספו לה ארבעה שדות חדשים בסופה. השדות האלה מספקים תמיכה לגופנים מסוג TrueType (להם תכונות משופרות לשינוי קנה המידה). לפניך השדות החדשים שנוספו ל-NEWTEXTMETRIC:

```
DWORD ntmFlags; /* indicates type of font */
UINT ntmSizeEM; /* size of an em */
UINT ntmCellHeight; /* font height */
UINT ntmAvegWidth; /* average character width */
```

למטרות פרק זה, אין צורך בשדות אלה, אך הם עשויים להביא לך תועלת ביישומים שתכתוב. רצוי לעיין במדריכי API כדי לקבל מידע נוסף.

10.5 חישוב אורך המחרוזת

התווים בגופן הנוכחי אינם בגודל זהה, ולכן אין אפשרות לדעת מהו אורך המחרוזת ביחידות לוגיות, לפי מספר התווים שבמחרוזת. כלומר, התוצאה המוחזרת על ידי הפונקציה ()strlen חסרת משמעות לניהול הפלט אל חלון, מכיון שהתווים הם ברוחב משתנה. חלונות 9x מכילה את הפונקציה ()GetTextExtentPoint32 שפותרת בעיה זו. הגדרתה היא:

```
BOOL GetTextExtentPoint32(HDC hdc, LPCSTR lpszString,
                          int len, LPSIZE lpSize);
```

במקרה זה hdc היא הידית של התקן הפלט. lpszString מצביע אל המחרוזת שאת אורכה ברצונך לדעת, והשדה len מכיל את מספר התווים שמהם מורכבת המחרוזת. רוחבה וגובהה של המחרוזת מוחזרים, ביחידות לוגיות, במבנה SIZE, שהמצביע lpSize פונה אליו. מבנה SIZE מוגדר כך:

```
typedef struct tagSIZE {
  LONG cx;  /* width */
  LONG cy;  /* height */
} SIZE;
```

לאחר הפעלת הפונקציה ()GetTextExtentPoint32, יכיל השדה cx את אורך המחרוזת. מסיבה זאת ניתן להשתמש בערך שמכיל שדה זה לקביעת נקודת ההתחלה של המחרוזת הבאה המיועדת להצגה, אם ברצונך להציג אותה בהמשך לפלט הקודם.

10.6 כיצד להשיג את נתוני מידות המערכת

חלונות מנהלת ומתרגמת באופן אוטומטי קואורדינטות לוגיות לפיקסלים, אך לעיתים תרצה לדעת מהו גודל התצוגה בפועל, במחשב שבו אתה מריץ את היישום שלך. כדי להשיג נתונים אלה ומידע נוסף, השתמש בפונקציה ()GetSystemMetrics, שהגדרתה היא:

```
int GetSystemMetrics(int what);
```

בהגדרה זו עליך לציין בשדה what איזה ערך ברצונך להשיג. הפונקציה ()GetSystemMetrics מסוגלת להשיג עבורך 39 ערכים שונים. הערכים לקואורדינטות של המסך מוחזרים בפיקסלים. בטבלה 10.3 מוצגות פקודות מאקרו עבור כמה מן הערכים הנפוצים ביותר:

טבלה 10.3: הצגת פקודות מאקרו עבור כמה מן הערכים הנפוצים ביותר

הערך	המידה שהוא מחזיר
SM_CXFULLSCREEN	רוחב שטח הלקוח כשהוא מוגדל
SM_CYFULLSCREEN	גובה שטח הלקוח כשהוא מוגדל
SM_CXICON	רוחב סמל גדול
SM_CYICON	גובה סמל גדול
SM_CXSMICON	רוחב סמל קטן
SM_CYSMICON	גובה סמל קטן
SM_CXSCREEN	רוחב המסך בכללותו
SM_CYSCREEN	גובה המסך בכללותו

10.7 הדגמה קצרה של פלט טקסט

לאחר שלמדת על חלק מפונקציות הטקסט של חלונות 9x, בוודאי תמצא תועלת בהדגמה קצרה של המאפיינים אלה. הקוד השלם של התוכנית **TextOut** נמצא בתיקיה Chap10\TextOut. כל פעם שתבחר את האפשרות !Text מהתפריט הראשי, תגרום להצגת מספר שורות טקסט. הטקסט יופיע בשחור על רקע טורקיז. הפלט לדוגמה מופיע בתרשים 10.1

הצגנו פה רק את קטע התוכנית של פונקצית ההודעות.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam )
{
   HDC hDC;
   TEXTMETRIC tm;
   SIZE size;

   switch( uMsg )
   {
     case WM_CREATE:
         /* get screen coordinates */
         maxX = GetSystemMetrics(SM_CXSCREEN);
         maxY = GetSystemMetrics(SM_CYSCREEN);
         break;
     case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                   hDC = GetDC(hWnd); /* get device context */

                   /* set text color to black */
                   SetTextColor(hDC, RGB(0, 0, 0));

                   /* set background color to turquoise */
                   SetBkColor(hDC, RGB(45, 135, 25));

                   /* get text metrics */
                   GetTextMetrics(hDC, &tm);

                   sprintf(str, "The font is %ld pixels
                          high.", tm.tmHeight);
```

```
        TextOut(hDC, X, Y, str,strlen(str)); /*
                output string */
        Y = Y + tm.tmHeight + tm.tmExternalLeading;
                /* next line */

        strcpy(str, "This is on the next line. ");
        TextOut(hDC, X, Y, str, strlen(str));
                /* output string */

        /* compute length of a string */
        GetTextExtentPoint32(hDC, str, strlen(str),
                            &size);
        sprintf(str, "Previous string is %ld units
                long", size.cx);
        X = size.cx; /* advance to end of previous
                        string */
        TextOut(hDC, X, Y, str, strlen(str));
        Y = Y + tm.tmHeight + tm.tmExternalLeading;
               /* next line */
        X = 0; /* reset X */

        sprintf(str, "Screen dimensions: %d %d",
                maxX, maxY);
        TextOut(hDC, X, Y, str, strlen(str));
        Y = Y + tm.tmHeight + tm.tmExternalLeading;
              /* next line */
        ReleaseDC(hWnd, hDC); /* Release DC */
        break;

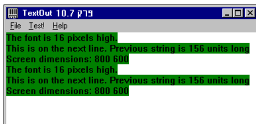
    case IDM_ABOUT :
            DialogBox( hInst, "AboutBox", hWnd,
             (DLGPROC)About );
            break;

    case IDM_EXIT :
            DestroyWindow( hWnd );
            break;
}
break;
```

```
      case WM_DESTROY :
              PostQuitMessage(0);
              break;

      default :
           return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
   }

   return( 0L );
}
```



תרשים 10.1: פלט לדוגמה מהתוכנית להצגת טקסט

כאשר נוצר החלון לראשונה, מתקבלת הודעת WM_CREATE והמספרים השלמים הגלובליים maxX ו-maxY מאותחלים לפי הקואורדינטות של המסך, באמצעות הפונקציה GetSystemMetrics(). ערכים אלה אינם משרתים כל מטרה בתוכנית זו, אך הם ישמשו אותנו בדוגמאות שבהמשך.

שים לב שהתוכנית מכריזה על שני משתנים גלובליים, X ו-Y, ומאתחלת את שניהם בערך 0. משתנים אלה יכילו את המיקום הנוכחי שבו יוצג הטקסט בחלון. התוכנית תעדכן את הערכים השמורים בהם באופן שוטף לאחר כל פעולת פלט.

החלק המעניין בתוכנית נמצא ברובו בהודעת WM_COMMAND. נבחן את ההודעה בפירוט, נתחיל עם משפט case IDM_TEST. כל פעם שמתקבלת הודעת IDM_TEST, מושג הֶקשר של התקן. לאחר מכן נקבע צבע הטקסט לשחור, וצבע הרקע נקבע לטורקיז. הקשר ההתקן מושג כל פעם שמתקבלת הודעת IDM_TEST, ולכן יש להגדיר את הצבעים כל פעם מחדש. כלומר, לא ניתן להגדירם פעם אחת בלבד (למשל, בתחילת התוכנית).

לאחר שנקבעו הצבעים, מושגים נתוני המידות של הטקסט. עתה, מתבצע פלט של השורה הראשונה. שים לב שהיא נבנית בעזרת הפונקציה sprintf(), ולאחר מכן נשלחת כפלט באמצעות הפונקציה TextOut(). פונקציית API, ופונקציית TextOut(), לא עורכות טקסט. עיצוב הפלט נתון בידיך, כמתכנת, ועליך לבנותו, ולאחר מכן להציגו באמצעות TextOut(). לאחר שהמחרוזת הוצגה, מקודמת קואורדינטת Y לשורה הבאה על ידי החלת הנוסחה שפיתחת קודם.

התוכנית ממשיכה ומוציאה כפלט את השורה: "This is on the next line". לפני שתימחק השורה כתוצאה מהקריאה הבאה לפונקציה ()sprintf, מחושב אורך המחרוזת באמצעות קריאה לפונקציה ()GetTextExtentPoint32. הערך המתקבל משמש לקידום קואורדינטת X לסוף השורה הקודמת, לפני הדפסת השורה הבאה. שים לב שכאן, קואורדינטת Y אינה משתנה. דבר זה גורם להצגת המחרוזת הבאה מייד לאחר המחרוזת הקודמת. לפני שהיא ממשיכה, מקדמת התוכנית את Y לשורה הבאה, ומאפסת את X, כלומר, ממקמת אותו בקואורדינטה השמאלית הקיצונית. דבר זה גורם להצגת קטע הפלט הבא בתחילת השורה הבאה.

לבסוף, מוצגים מימדי המסך כולו וקואורדינטת Y מקודמת לשורה הבאה. כל פעם שאתה בוחר באפשרות !Test, מוצג הטקסט במיקום נמוך יותר בחלון, וכך לא נמחק הטקסט הקודם. כתוצאה מכך, מוצגת כל קבוצת שורות מתחת לקבוצה קודמת.

10.8 פתרון בעיית הצביעה מחדש (Repaint)

בעזרת התוכנית הקודמת הדגמנו כמה מפונקציות הטקסט והמערכת, אך גם נתקלנו שוב בבעיה בסיסית, שנדונה לראשונה בפרק 3. הבעיה היא, שהטקסט הולך לאיבוד כאשר אתה מריץ את התוכנית, מציג טקסט ולאחר מכן מפעיל חלון אחר המסתיר את החלון הנוכחי. כשתשוב ותחשוף את החלון, תגלה שחלק הטקסט שהוסתר על ידי החלון נעלם. הסיבה לכך היא שכל תוכנית צריכה לצבוע מחדש את החלון שלה כשהיא מקבלת הודעת WM_PAINT, אך התוכנית הקודמת לא עושה זאת. דבר זה מעלה את השאלה הגדולה: באיזה מנגנון יש להשתמש כדי לשחזר תוכן של חלון שהוסתר? כפי שהזכרנו קודם, קיימות שלוש שיטות בסיסיות: אפשר להפיק מחדש את הפלט אם הוא מתהווה באמצעות שיטות חישוב כלשהי. שנית, אפשר לשמור עותק מכל אירוע תצוגה, ולהציג לאחר מכן "הקלטה" שלו. שלישית, תוכל לנהל חלון וירטואלי ולהעתיק את תוכנו אל החלון הממשי כל פעם שתתקבל הודעת WM_CREATE. השיטה הרווחת ביותר היא כמובן השלישית, ובה נעסוק בפרק זה. כפי שיתברר לך, חלונות מספקת תמיכה ניכרת לשיטה זו.

10.9 רעיון החלון הווירטואלי

תיאור באופן כללי, כיצד יתבצע פלט באמצעות **חלון וירטואלי** (Virtual Window).

תחילה, יווצר הקשר וירטואלי של התקן, אשר יהיה תואם לֶהקשר ההֶתקן הממשי. לאחר מכן, ייכתב כל הפלט אל הקשר ההתקן הווירטואלי. כל פעם שתתקבל הודעת WM_PAINT, תוכן הקשר ההתקן הווירטואלי (החלון הווירטואלי) יועתק אל הקשר ההתקן הממשי, ויגרום להצגת הפלט בחלון שעל המסך. כך תמיד ישמר רישום של התכולה הנוכחית של החלון. אם אותו חלון יוסתר, ולאחר מכן ישוב וייחשף, תתקבל הודעת WM_PAINT ותוכן החלון ישוחזר באופן אוטומטי.

10.10 פונקציות API נוספות

כדי ליצור חלון וירטואלי ולהשתמש בו, עליך להשתמש בכמה פונקציות API. בארבע מהן כבר עסקנו: ()CreateCompatibleDC, ()SelectObject, ()GetStockObject וב-()BitBlt. בנוסף, נשתמש בפונקציות ()CreateCompatibleBitmap ו-()PatBlt, שיתוארו שוב בקצרה, לרענון.

הפונקציה ()CreateCompatibleBitmap יוצרת מפת-סיביות שתואמת את הקשר ההתקן שצוין. מפה זו יכולה לשמש כל הקשר התקן השמור בזיכרון (שנוצר באמצעות הפונקציה ()CreateCompatibleDC), בתנאי שהוא תואם את הקשר ההתקן שצוין. לפניך הגדרתה הכללית:

```
HBITMAP CreateCompatibleBitmap(HDC hdc, int width, int height);
```

בדוגמה, hdc היא הידית להקשר ההתקן שמפת-הסיביות תתאים לו. מימדי המפה מוגדרים על ידי הערכים בפרמטרים width ו-height. ערכים אלה נתונים בפיקסלים. הפונקציה מחזירה ידית למפת-הסיביות התואמת, או ערך NULL במקרה של כשל.

הפונקציה ()PatBlt ממלאת שטח מלבני בצבע ובדוגמה של **המברשת** (Brush) הנוכחית. מברשת היא עצם המגדיר כיצד ימולא חלון (או אזור כלשהו). מילוי של שטח בעזרת מברשת הוא פעולה המכונה בדרך כלל **צביעה** (Painting). לפניך הגדרתה הכללית של הפונקציה ()PatBlt:

```
BOOL PatBlt(HDC hdc, int X, int Y, int width,
                         int height, DWORD dwRaster);
```

במקרה זה, hdc היא הידית של ההתקן שיש למלא. הקואורדינטות X ו-Y מגדירות את הפינה השמאלית-העליונה של השטח שיש למלא. רוחבו וגובהו של השטח מוגדרים לפי הערכים שמכילים הפרמטרים width ו-height. הערך המועבר בשדה dwRaster קובע איזה שימוש ייעשה במברשת. הוא חייב להכיל, כערך, אחת מפקודות המאקרו שלהלן:

dwRaster	משמעות
BLACKNESS	האזור מופיע בשחור (התוכנית מתעלמת מהמברשת)
WHITENESS	האזור מופיע בלבן (התוכנית מתעלמת מהמברשת)
PATCOPY	המברשת מועתקת אל האזור
PATINVERT	פעולת OR בין המברשת לאזור
DSTINVERT	באזור מתבצע היפוך (התוכנית מתעלמת מהמברשת)

לכן, אם ברצונך להחיל את המברשת הנוכחית ללא שינוי, בחר בערך PATCOPY לפרמטר dwRaster. הפונקציה תחזיר ערך לא-אפס אם סיימה בהצלחה, וערך אפס אם לא.

כעת, כשידוע לך אילו פונקציות ישמשו אותך, תוכל לראות כיצד ליצור חלון וירטואלי.

10.11 יצירת חלון וירטואלי והשימוש בו

התחל בחזרה על הנוהל שיושם. כדי ליצור אמצעי פשוט ונוח לשחזור תכולת חלון לאחר שהתקבלה הודעת WM_PAINT, תנהל התוכנית חלון וירטואלי וכל הפלט ייכתב אל אותו חלון. כל פעם שמתקבלת בקשה לצביעה מחדש, תועתק תכולת החלון הווירטואלי אל החלון המופיע פיסית על המסך. ייִשם שיטה זו כעת.

יצירת החלון הווירטואלי

ראשית, יש ליצור הקשר התקן וירטואלי, אשר תואם את הקשר ההתקן הנוכחי. דבר שיתבצע פעם אחת בלבד, כאשר התוכנית מתחילה לרוץ, בעת שמתקבלת הודעת WM_CREATE. אותו הקשר התקן תואם ימשיך להתקיים כל זמן שהתוכנית רצה. לפניך הקוד המבצע פונקציה זו:

```
case WM_CREATE:
        /* get screen coordinates */
        maxX = GetSystemMetrics(SM_CXSCREEN);
        maxY = GetSystemMetrics(SM_CYSCREEN);
                          /* make a compatible memory image */
        hDC = GetDC(hWnd);
        memdc = CreateCompatibleDC(hDC);
        hBit = CreateCompatibleBitmap(hDC, maxX, maxY);
        SelectObject(memdc, hBit);
        hBrush = GetStockObject(WHITE_BRUSH);
        SelectObject(memdc, hBrush);
        PatBlt(memdc, 0, 0, maxX, maxY, PATCOPY);
        ReleaseDC(hWnd, hDC);
  break;
```

הבה נבחן את הקוד הזה מקרוב. ראשית, הפונקציה משיגה את מימדי המסך. נתון זה ישמש ליצירת מפת-סיביות תואמת. לאחר מכן, מושג הקשר ההתקן הנוכחי. הפונקציה יוצרת הקשר התקן תואם בזיכרון בעזרת הפונקציה CreateCompatibleDC(). הידית להקשר ההתקן נשמרת בשדה memdc, שהוא משתנה גלובלי. נוצרת מפת-סיביות תואמת. כך נוצר מיפוי ביחס של אחד-לאחד בין החלון הווירטואלי לחלון הפיסי. מימדי מפת-הסיביות הם אלה של גודל המסך המירבי. דבר המבטיח שמפת-הסיביות תהיה תמיד גדולה די הצורך, לשחזור מלא של החלון ללא תלות בגודלו (למעשה, ניתן היה להשתמש בערכים נמוכים מעט יותר, הואיל וקווי הגבול של החלון אינם נצבעים מחדש, שיפור פעוט זה נתון בידך, כתרגיל).

הידית למפת-הסיביות נשמרת במשתנה הגלובלי hbit. עתה מושגת ממלאי המערכת מברשת לבנה, והידית שלה נשמרת במשתנה הגלובלי hbrush. המברשת מוצבת בהקשר ההתקן שבזיכרון, ולאחר מכן הפונקציה PatBlt() צובעת את כל החלון

הווירטואלי בעזרתה. כך, החלון הווירטואלי יקבל רקע לבן, התואם את הרקע של החלון הפיסי (זכור, הצבעים האלה בשליטתך, הצבעים שאנו משתמשים כאן מקריים). לבסוף, התוכנית משחררת את הקשר ההתקן הפיסי, בעוד שהקשר ההתקן שבזיכרון ממשיך להתקיים עד שהתוכנית מסתיימת.

כיצד להשתמש בחלון הווירטואלי

לאחר שהחלון הווירטואלי נוצר, כל הפלט ינותב אליו (המקרה היחיד שבו הפלט מנותב בפועל לחלון הפיסי הוא קבלת הודעת WM_PAINT). לפניך גירסה מתוקנת של הודעת IDM_ TEST אשר משתמשת בטכניקת החלון הווירטואלי:

```
case IDM_TEST :
   /* set text color to black */
   SetTextColor(memdc, RGB(0, 0, 0));

   /* set background mode to transparent */
   SetBkMode(memdc, TRANSPARENT);

   /* get text metrics */
   GetTextMetrics(memdc, &tm);

   sprintf(str, "The font is %ld pixels high.", tm.tmHeight);
   TextOut(memdc, X, Y, str, strlen(str)); /* output string */
   Y = Y + tm.tmHeight + tm.tmExternalLeading; /* next line */

   strcpy(str, "This is on the next line. ");
   TextOut(memdc, X, Y, str, strlen(str)); /* output string */

   /* compute length of a string */
   GetTextExtentPoint32(memdc, str, strlen(str), &size);
   sprintf(str, "Previous string is %ld units long", size.cx);
   X = size.cx; /* advance to end of previous string */
   TextOut(memdc, X, Y, str, strlen(str));
   Y = Y + tm.tmHeight + tm.tmExternalLeading; /* next line */
   X = 0; /* reset X */

   sprintf(str, "Screen dimensions: %d %d", maxX, maxY);
   TextOut(memdc, X, Y, str, strlen(str));
   Y = Y + tm.tmHeight + tm.tmExternalLeading; /* next line */
   InvalidateRect(hWnd, NULL, 1);
   break;
```

גירסה זו מנתבת את כל הפלט לשדה memdc ומפעילה את הפונקציה ()InvalidateRect כדי לגרום לעדכון החלון הפיסי.

שים לב שגירסה זו גם קובעת את מצב הרקע ומציבה אותו על TRANSPARENT. הדבר גורם להצגת הטקסט בחלון ללא כל שינוי בצבע הרקע.

בכל פעם שמתקבלת הודעת WM_PAINT, מועתק תוכן החלון הווירטואלי אל ההתקן הפיסי, באמצעות קטע הקוד הבא:

```
case WM_PAINT: /* process a repaint request */
hDC = BeginPaint(hWnd, &paintstruct); /* get DC */
   /* now, copy memory image onto screen */
   BitBlt(hDC, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
   EndPaint(hWnd, &paintstruct); /* release DC */
break;
```

כפי שאתה רואה, הפונקציה ()BitBlt משמשת כאן להעתקת התמונה מ- memdc ל-hDC. זכור, הפרמטר SRCCOPY מעתיק את התמונה כמות שהיא, בלא שינוי, מהמקור אל היעד. כל הפלט נשמר ב-memdc, ולכן משפט זה גורם להצגת הפלט בפועל. אם החלון הוסתר ולאחר מכן נחשף מחדש, תתקבל הודעת WM_PAINT, ותכולת החלון תשוחזר באופן אוטומטי.

יש דרכים רבות לגשת לשחזור תוכן חלון. השיטה שתוארה ישימה בקשת רחבה של נסיבות, ובדרך כלל פועלת ביעילות. התוכנית שלך מקבלת את הקואורדינטות של האזור שיש לצובעו מחדש, ולכן תוכל לייעל באופן ממשי את השיגרה הקודמת על ידי שחזור ברירני של אותו חלק חלון שנהרס (נסה להכניס שיפור זה בעצמך).

התוכנית השלמה לחלונות וירטואליים

לפניך התוכנית השלמה **VirtualWin** המדגימה את נושא החלונות הווירטואליים (התוכנית בשלמותה נמצאת בתיקיה: Chap10\VirtualWin):

```
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include "VirtualWin.h"

/* Demonstrate a Message Box. */

#if defined (WIN32)
                        #define IS_WIN32 TRUE
#else
                        #define IS_WIN32 FALSE
#endif
```

```
#define IS_NT      IS_WIN32 && (BOOL)(GetVersion() < 0x80000000)
#define IS_WIN32S  IS_WIN32 && (BOOL)(!(IS_NT) &&
                   (LOBYTE(LOWORD(GetVersion()))<4))
#define IS_WIN95   (BOOL)(!(IS_NT) && !(IS_WIN32S)) && IS_WIN32

HINSTANCE hInst;  // current instance
BOOL RegisterWin95( CONST WNDCLASS* lpwc );

LPCTSTR lpszAppName  = "MyApp";
LPCTSTR lpszTitle    = " VirtualWin  פרק 10.11";
char str[255]; /* holds output strings */

int X=0, Y=0; /* current output location */
int maxX, maxY; /* screen dimensions */

int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                  hPrevInstance,LPTSTR lpCmdLine, int nCmdShow)
{
   MSG      msg;
   HWND     hWnd;
   WNDCLASS wc;

    // Register the main application window class.
   //..........................................
   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = (WNDPROC)WndProc; /* window function */
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInstance;  /* handle to this instance */
   wc.hIcon         = LoadIcon( hInstance, lpszAppName );
                      /* icon style */
   wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
                      /* cursor style */
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName; /* window class name */

   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );
```

```
    hInst = hInstance;

   // Create the main application window.
   //...................................
   hWnd = CreateWindow( lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                      );

   if ( !hWnd )
      return( FALSE );

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );

   while( GetMessage( &msg, NULL, 0, 0) )
   {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }

   return( msg.wParam );

}

BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style          = lpwc->style;
   wcex.lpfnWndProc    = lpwc->lpfnWndProc;
   wcex.cbClsExtra     = lpwc->cbClsExtra;
   wcex.cbWndExtra     = lpwc->cbWndExtra;
   wcex.hInstance      = lpwc->hInstance;
   wcex.hIcon          = lpwc->hIcon;
```

```
   wcex.hCursor       = lpwc->hCursor;
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;

   // Added elements for Windows 95.
   //..............................
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");

   return RegisterClassEx( &wcex );
}

/* This function is called by Windows 95 and is passed
   messages from the message queue.*/
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam )
{
   HDC hDC;
   PAINTSTRUCT paintstruct;
   TEXTMETRIC tm;
   SIZE size;

   static HDC memdc; /* store the virtual device handle */
   static HBITMAP hBit; /* store the virtual bitmap */
   static HBRUSH hBrush; /* store the brush handle */

   switch( uMsg )
   {
      case WM_CREATE:
         /* get screen coordinates */
         maxX = GetSystemMetrics(SM_CXSCREEN);
         maxY = GetSystemMetrics(SM_CYSCREEN);
         /* make a compatible memory image */
         hDC = GetDC(hWnd);
         memdc = CreateCompatibleDC(hDC);
         hBit = CreateCompatibleBitmap(hDC, maxX, maxY);
         SelectObject(memdc, hBit);
         hBrush = (HBRUSH)GetStockObject(WHITE_BRUSH);
         SelectObject(memdc, hBrush);
         PatBlt(memdc, 0, 0, maxX, maxY, PATCOPY);
         ReleaseDC(hWnd, hDC);
         break;
```

```
    case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
               case IDM_TEST :
                  /* set text color to black */
                  SetTextColor(memdc, RGB(0, 0, 0));

                  /* set background mode to transparent */
                  SetBkMode(memdc, TRANSPARENT);

                  /* get text metrics */
                  GetTextMetrics(memdc, &tm);

                  sprintf(str, "The font is %ld pixels high."
                          , tm.tmHeight);
                  TextOut(memdc, X, Y, str, strlen(str));
                          /* output string */
                  Y = Y + tm.tmHeight + tm.tmExternalLeading;
                          /* next line */

                  strcpy(str, "This is on the next line. ");
                  TextOut(memdc, X, Y, str, strlen(str));
                         /* output string * /

                  /* compute length of a string */
                  GetTextExtentPoint32(memdc, str, strlen(str)
                                        , &size);
                  sprintf(str, "Previous string is %ld units
                          long", size.cx);
                  X = size.cx; /* advance to end of previous
                                  string */
                  TextOut(memdc, X, Y, str, strlen(str));
                  Y = Y + tm.tmHeight + tm.tmExternalLeading;
                          /* next line */
                  X = 0; /* reset X */

                  sprintf(str, "Screen dimensions: %d %d",
                          maxX, maxY);
                  TextOut(memdc, X, Y, str, strlen(str));
                  Y = Y + tm.tmHeight + tm.tmExternalLeading;
                         /* next line */
                  InvalidateRect(hWnd, NULL, 1);
                  break;
```

```
                case IDM_ABOUT :
                        DialogBox( hInst, "AboutBox", hWnd,
                                    (DLGPROC)About );
                        break;

                case IDM_EXIT :
                        DestroyWindow( hWnd );
                        break;
             }
             break;
      case WM_PAINT: /* process a repaint request */
           hDC = BeginPaint(hWnd, &paintstruct); /* get DC */

           /* now, copy memory image onto screen */
           BitBlt(hDC, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
           EndPaint(hWnd, &paintstruct); /* release DC */
           break;
      case WM_DESTROY :
           DeleteDC(memdc); /* delete the memory device */
           PostQuitMessage(0);
           break;
      default :
        return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
   }

   return( 0L );
}

LRESULT CALLBACK About( HWND hDlg,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
   switch (message)
   {
       case WM_INITDIALOG:
               return (TRUE);
```

```
case WM_COMMAND:
               if (   LOWORD(wParam) == IDOK
                   || LOWORD(wParam) == IDCANCEL)
               {
                       EndDialog(hDlg, TRUE);
                       return (TRUE);
               }
               break;
   }

   return (FALSE);
}
```

כשתריץ את התוכנית, תיווכח בשיפור מיידי. ראשית, כל פעם שתסתיר ולאחר מכן תשוב ותחשוף חלון מסוים, תכולתו תשוחזר.

10.12 כיצד לשנות גופנים

כפי שבוודאי ידוע לך, אחת המטרות העיקריות של חלונות 9x היא לאפשר שליטה מלאה על ממשק המשתמש. לפיכך היא מכילה קשת רחבה ומגוונת של מאפיינים מבוססי-טקסט, לשימושך. אחד המאפיינים רבי העוצמה ביותר הוא אוסף **הגופנים** (Fonts). כאשר תשתמש בחלונות, תוכל לבחור מכמה גופני מערכת. תוכל גם ליצור גופנים אישיים משלך. בסעיפים הבאים נעסוק בנושאים אלה.

הגופנים המובנים במערכת

גופני המערכת (Built-In Fonts) הם עצמים המובנים במערכת, שניתן לבחור בהם באמצעות הפונקציה ()GetStockObject. בעת כתיבת ספר זה, חלונות תומכת בשבעה גופנים מובנים. פקודות המאקרו הקשורות בגופנים אלה מובאות לפניך:

שם המאקרו	תיאור הגופן
ANSI_FIXED_FONT	גופן בעל מרווחים קבועים
ANSI_VAR_FONT	גופן בעל מרווחים משתנים
DEVICE_DEFAULT_FONT	גופן ברירת המחדל של ההתקן
DEFAULT_GUI_FONT	Default GUI font
OEM_FIXED_FONT	OEM-defined font
SYSTEM_FONT	גופן המערכת של חלונות 9x
SYSTEM_FIXED_FONT	גופן המערכת של גרסאות קודמות של חלונות

גופני המערכת הם גופני תווים המשמשים את חלונות לכיתוב בתפריטים ובתיבות דו-שיח. גרסאות קודמות של חלונות השתמשו בגופן מערכת בעל מרווחים קבועים בין התווים (Fixed Pitch Font), אבל בגרסאות המתקדמות יותר, מ-Windows 3.x ואילך, גופן המערכת הוא בעל מרווחים משתנים.

הבחירה והשימוש בגופן מובנה פשוטים מאוד. התוכנית שלך תצטרך ליצור ידית מסוג HFONT לגופן. לאחר מכן עליה לטעון את הגופן הרצוי בעזרת הפונקציה ()GetStockObject, המחזירה ידית לגופן. כדי לעבור לגופן הנבחר, בחר אותו על ידי הפונקציה ()SelectObject, עם הגופן החדש כפרמטר. הפונקציה תחזיר ידית לגופן הקודם (שאולי תרצה לשמור, ולחזור אליו לאחר סיום השימוש בגופן שנבחר).

התוכנית **Fonts** שלפניך מדגימה את שינוי הגופנים. היא מוסיפה לתפריט אפשרות חדשה - Font. כל פעם שתבחר באפשרות זו, ייבחר לסירוגין (במתכונת של מפסק) גופן ברירת המחדל של המערכת, או גופן ANSI המשתנה.

```
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include "Fonts.h"

/* Demonstrate a Message Box. */

#if defined (WIN32)
        #define IS_WIN32 TRUE
#else
        #define IS_WIN32 FALSE
#endif

#define IS_NT       IS_WIN32 && (BOOL)(GetVersion() < 0x80000000)
#define IS_WIN32S   IS_WIN32 && (BOOL)(!(IS_NT) &&
                    (LOBYTE(LOWORD(GetVersion()))<4))
#define IS_WIN95    (BOOL)(!(IS_NT) && !(IS_WIN32S)) && IS_WIN32

HINSTANCE hInst;  // current instance
BOOL RegisterWin95( CONST WNDCLASS* lpwc );

LPCTSTR lpszAppName  = "MyApp";
LPCTSTR lpszTitle    = " Fonts 10.12 פרק";
char str[255]; /* holds output strings */

int X=0, Y=0; /* current output location */
int maxX, maxY; /* screen dimensions */
int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                  hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
```

```
{
   MSG       msg;
   HWND      hWnd;
   WNDCLASS wc;

   // Register the main application window class.
   //.........................................
   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = (WNDPROC)WndProc; /* window function */
   wc.cbClsExtra     = 0;
   wc.cbWndExtra     = 0;
   wc.hInstance      = hInstance;  /* handle to this instance */
   wc.hIcon          = LoadIcon( hInstance, lpszAppName );
                       /* icon style */
   wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
                       /* cursor style */
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName; /* window class name */

   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );

   hInst = hInstance;

   // Create the main application window.
   //...................................
   hWnd = CreateWindow( lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                       );
```

```
   if ( !hWnd )
      return( FALSE );

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );

   while( GetMessage( &msg, NULL, 0, 0) )
   {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }

   return( msg.wParam );
}

BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style         = lpwc->style;
   wcex.lpfnWndProc   = lpwc->lpfnWndProc;
   wcex.cbClsExtra    = lpwc->cbClsExtra;
   wcex.cbWndExtra    = lpwc->cbWndExtra;
   wcex.hInstance     = lpwc->hInstance;
   wcex.hIcon         = lpwc->hIcon;
   wcex.hCursor       = lpwc->hCursor;
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;

   // Added elements for Windows 95.
   //..............................
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");

   return RegisterClassEx( &wcex );
}
```

```
/* This function is called by Windows 95 and is passed
   messages from the message queue.*/
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam )
{
   HDC hDC;
   PAINTSTRUCT paintstruct;
   static TEXTMETRIC tm;
   SIZE size;
   static fontswitch = 0;

   static HDC memdc; /* store the virtual device handle */
   static HBITMAP hBit; /* store the virtual bitmap */
   static HBRUSH hBrush; /* store the brush handle */
   static HFONT holdf, hnewf; /* store the font handles */

   switch( uMsg )
   {
      case WM_CREATE:      /* get screen coordinates */
        maxX = GetSystemMetrics(SM_CXSCREEN);
        maxY = GetSystemMetrics(SM_CYSCREEN);

        /* make a compatible memory image device */
        hDC = GetDC(hWnd);
        memdc = CreateCompatibleDC(hDC);
        hBit = CreateCompatibleBitmap(hDC, maxX, maxY);
        SelectObject(memdc, hBit);
        hBrush = (HBRUSH)GetStockObject(WHITE_BRUSH);
        SelectObject(memdc, hBrush);
        PatBlt(memdc, 0, 0, maxX, maxY, PATCOPY);

        /* get new font */
        hnewf = (HFONT)GetStockObject(ANSI_VAR_FONT);

        ReleaseDC(hWnd, hDC);
        break;
```

```
    case WM_COMMAND :
      switch( LOWORD( wParam ) )
       {
         case IDM_TEST :
        /* set text color to black and mode to transparent */
        SetTextColor(memdc, RGB(0, 0, 0));
        SetBkMode(memdc, TRANSPARENT);

        /* get text metrics */
        GetTextMetrics(memdc, &tm);

        sprintf(str, "The font is %ld pixels high.",
                tm.tmHeight);
        TextOut(memdc, X, Y, str, strlen(str));
            /* output string */
        Y = Y + tm.tmHeight + tm.tmExternalLeading;
            /* next line */

        strcpy(str, "This is on the next line. ");
        TextOut(memdc, X, Y, str, strlen(str));
            /* output string */

        /* compute length of a string */
        GetTextExtentPoint32(memdc, str, strlen(str), &size);
        sprintf(str, "Previous string is %ld units long",
                size.cx);
        X = size.cx; /* advance to end of previous string */
        TextOut(memdc, X, Y, str, strlen(str));
        Y = Y + tm.tmHeight + tm.tmExternalLeading;
            /* next line */
        X = 0; /* reset X */

        sprintf(str, "Screen dimensions: %d %d", maxX, maxY);
        TextOut(memdc, X, Y, str, strlen(str));
        Y = Y + tm.tmHeight + tm.tmExternalLeading;
            /* next line */
        InvalidateRect(hWnd, NULL, 1);
        break;
```

```
                case IDM_CHANGE_FONT:
                    if(!fontswitch) {
                    /* switch to new font */
                       holdf = (HFONT)SelectObject(memdc,
                       hnewf);
                       fontswitch = 1;
                          }
                    else { /* switch to old font */
                       SelectObject(memdc, holdf);
                       fontswitch = 0;
                          }
                       break;
                case IDM_ABOUT :
                      DialogBox( hInst, "AboutBox", hWnd,
                                (DLGPROC)About );
                      break;

                case IDM_EXIT :
                      DestroyWindow( hWnd );
                      break;
              }
              break;
    case WM_PAINT: /* process a repaint request */
      hDC = BeginPaint(hWnd, &paintstruct); /* get DC */

      /* now, copy memory image onto screen */
      BitBlt(hDC, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
      EndPaint(hWnd, &paintstruct); /* release DC */
      break;
      case WM_DESTROY :
              DeleteDC(memdc); /* delete the memory device */
              PostQuitMessage(0);
              break;

      default :
        return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
   }

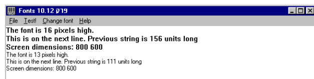
   return( 0L );
}
```

```
LRESULT CALLBACK About( HWND hDlg,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
   switch (message)
   {
       case WM_INITDIALOG:
               return (TRUE);

       case WM_COMMAND:
               if (   LOWORD(wParam) == IDOK
                   || LOWORD(wParam) == IDCANCEL)
               {
                       EndDialog(hDlg, TRUE);
                       return (TRUE);
               }
               break;
   }

   return (FALSE);
}
```

התוכנית השלמה נמצאת בתיקיה Chap10\Fonts.



תרשים 10.2: פלט לדוגמה מהתוכנית **Fonts**.

כיצד ליצור גופנים אישיים

אף שהדבר אולי נראה מסובך, למעשה מאוד פשוט ליצור גופן אישי. יצירת גופנים אישיים מעניקה לך שני יתרונות בולטים. ראשית, גופן אישי מעניק ליישום שלך מראה ייחודי שיבדיל אותו מכל שאר היישומים. שנית, יצירת גופן מאפשרת לך לשלוט בדיוק רב במה שקורה כאשר הטקסט יוצא כפלט. לפני שתתחיל, חשוב שתבין **שאינך עומד להגדיר** גופן חדש, אלא **להתאים** גופן קיים כך שיענה בדיוק על דרישותיך (כלומר, אינך צריך להגדיר את הצורה של כל תו בגופן שאתה יוצר).

כדי להגדיר גופן משלך, השתמש בפונקציה ()CreateFont, שהגדרתה הכללית היא:

```
HFONT CreateFont(int Height, int Width, int Escapement,
          int Orientation, int Weight, DWORD Ital,
          DWORD Underline, DWORD StrikeThru,
          DWORD Charset, DWORD Precision,
          DWORD ClipPrecision, DWORD Quality,
          DWORD Pitch, LPCSTR FontName);
```

גובה הגופן מועבר בפרמטר Height, ורוחבו בפרמטר Width. אם Width מכיל אפס, חלונות תבחר ערך מתאים המבוסס על **היחס גובה/רוחב** (Aspect Ratio) הנוכחי. הערכים ב-Height וב-Width נקובים ביחידות לוגיות.

ניתן להוציא טקסט כפלט אל החלון בכל זווית רצויה. הזווית שבה יוצג הטקסט נקבעת על ידי הפרמטר Escapement. עבור טקסט אופקי רגיל, יכיל הפרמטר הזה ערך 0. אם לא, יצוין בפרמטר זה ערך במעלות שבו יש לסובב את הטקסט על צירו. ערך זה רשום ביחידות של עשירית המעלה, ולכן, לדוגמה, הערך 900 יגרום לסיבוב הטקסט 90 מעלות על צירו, כך שהפלט יהיה אנכי.

ניתן לקבוע את הזווית של כל תו בנפרד בעזרת הפרמטר Orientation. גם כאן יחידות המידה הן עשיריות המעלה, והן משמשות לציון הזווית של כל תו ביחס לקו האופקי.

הפרמטר Weight קובע את **השקלול המועדף** (Prefered Weight) של הגופן בטווח שבין 0 ל-1000. ערך 0 מייצג את שקלול ברירת המחדל. כדי להגדיר שקלול רגיל, השתמש בערך 400. להדגשת תווים, השתמש בערך 700. תוכל להשתמש גם בפקודות המאקרו המופיעות ברשימה הבאה לציון שקלול הגופן:

FW_DONTCARE
FW_THIN
RW_LIGHT
FW_NORMAL
FW_SMIBOLD
FW_BOLD
FW_EXTRABOLD
FW_HEAVY

כדי ליצור גופן **נטוי**, הקצה לפרמטר Ital ערך לא-אפס (בגופן רגיל, יכיל פרמטר זה ערך אפס). כדי ליצור גופן מודגש ב**קו תחתי**, קבע לפרמטר Underline ערך לא-אפס. כדי לקבל גופן ללא קו תחתי, קבע לפרמטר זה ערך אפס. ליצירת גופן "**מחוק**", קבע לפרמטר StrikeThru ערך לא-אפס. ליצירת גופן ללא קו מחיקה, קבע ערך אפס.

הפרמטר Charset קובע איזו סדרת תווים רצויה לך. בדוגמה הבאה נשתמש בסדרת ANSI_CHARSET. הפרמטר Precision מציין את **דרגת הדיוק** המועדפת של הקלט, כלומר איזו מידה של תאימות צריכה להתקיים בין מאפייני הפלט בפועל לבין מאפייני הגופן שביקשת. בדוגמה המופיעה בפרק נשתמש ב- OUT_DEFAULT_PREC. הפרמטר

ClipPrecision מגדיר את מידת הדיוק של הגזירה. המונח **נתוני גזירה** (Clipping Precision) מתייחס לאופן שבו כל תו החורג מגבולות הגזירה אמור "להיקטם". הערך המשמש אותנו בדוגמה הבאה הוא CLIP_DEFAULT_PRECIS (תוכל למצוא במדריך API שלך ערכים תקפים אחרים לפרמטרים Chartset, Precision ו-ClipPrecision).

הפרמטר quality קובע את דרגת ההתאמה בין הגופן הלוגי לבין הגופנים הפיסיים שהתוכנית מספקת בפועל להתקן הפלט שהוגדר. הפרמטר יכול להכיל אחד משלושה ערכים:

DEFAULT_QUALITY
DRAFT_QUALITY
PROFF_QUALITY

הפרמטר Pitch, **פסיעה**, מגדיר את המרווח בין התווים ואת המשפחה שאליה משתייך הגופן שנבחר. גם פרמטר זה יכול להכין אחד משלושה ערכים:

DEFAULT_PITCH
FIXED_PITCH
VARIABLE_PITCH

קיימות שש משפחות אפשריות של גופנים:

FF_DECORATIVE
FF_DONTCARE
FF_MODERN
FF_ROMAN
FF_SCRIPT
FF_SWISS

כדי לייצר ערך עבור הפרמטר Pitch, עליך להפעיל OR על אחד מערכי pitch ואחד הערכים של משפחת גופנים.

מצביע לשם הגופן מועבר באמצעות FontName. מותרים שמות באורך של עד 32 תווים. הגופן שאתה מגדיר חייב להיות מותקן במערכת שלך.

הפונקציה ()CreateFont מחזירה ידית לגופן אם סיימה בהצלחה, וערך NULL אם לא.

✍ **הערה:** חשוב להבין שמבחינה טכנית, הפונקציה ()CreateFont אינה יוצרת גופן חדש. היא רק מתאימה אותו במידת האפשר, על סמך המידע שאתה מספק למערכת, לגופנים הפיסיים הממשיים שיש במערכת.

חובה למחוק את הגופנים שנוצרו בעזרת הפונקציה ()CreateFont לפני סיום התוכנית. כדי למחוק גופן, הפעל את הפונקציה ()DeleteObject.

לפניך תוכנית **CustFont** המדגימה שני גופנים אישיים. הראשון מבוסס על גופן Courier New, השני על הגופן Century Gothic. כל פעם שתבחר את הפריט Change Font מתוך התפריט, ייבחר ויוצג גופן חדש.

התוכנית משתמשת בקובץ משאבים כמו התוכנית בסעיף הקודם.

```
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include "CustFont.h"

/* Demonstrate a Message Box. */

#if defined (WIN32)
                        #define IS_WIN32 TRUE
#else
                        #define IS_WIN32 FALSE
#endif

#define IS_NT       IS_WIN32 && (BOOL)(GetVersion() < 0x80000000)
#define IS_WIN32S  IS_WIN32 && (BOOL)(!(IS_NT) &&
                  (LOBYTE(LOWORD(GetVersion()))<4))
#define IS_WIN95    (BOOL)(!(IS_NT) && !(IS_WIN32S)) && IS_WIN32

HINSTANCE hInst;   // current instance
BOOL RegisterWin95( CONST WNDCLASS* lpwc );

LPCTSTR lpszAppName  = "MyApp";
LPCTSTR lpszTitle    = " Fonts 10.12 פרק";
char str[255]; /* holds output strings */

int X=0, Y=0; /* current output location */
int maxX, maxY; /* screen dimensions */

int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                  hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
{
   MSG       msg;
   HWND      hWnd;
   WNDCLASS wc;

   // Register the main application window class.
   //.............................................
   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = (WNDPROC)WndProc; /* window function */
   wc.cbClsExtra     = 0;
   wc.cbWndExtra     = 0;
```

```
   wc.hInstance     = hInstance;  /* handle to this instance */
   wc.hIcon         = LoadIcon( hInstance, lpszAppName );
                      /* icon style */
   wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
                      /* cursor style */
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName; /* window class name */

   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );

    hInst = hInstance;

   // Create the main application window.
   //..................................
   hWnd = CreateWindow( lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                      );

   if ( !hWnd )
      return( FALSE );

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );

   while( GetMessage( &msg, NULL, 0, 0) )
   {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }

   return( msg.wParam );
}
```

```
BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style         = lpwc->style;
   wcex.lpfnWndProc   = lpwc->lpfnWndProc;
   wcex.cbClsExtra    = lpwc->cbClsExtra;
   wcex.cbWndExtra    = lpwc->cbWndExtra;
   wcex.hInstance     = lpwc->hInstance;
   wcex.hIcon         = lpwc->hIcon;
   wcex.hCursor       = lpwc->hCursor;
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;

   // Added elements for Windows 95.
   //..............................
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");

   return RegisterClassEx( &wcex );
}

/* This function is called by Windows 95 and is passed
   messages from the message queue.*/
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam )
{
   HDC hDC;
   PAINTSTRUCT paintstruct;
   static TEXTMETRIC tm;
   SIZE size;
   static fontswitch = 0;

   static HDC memdc; /* store the virtual device handle */
   static HBITMAP hBit; /* store the virtual bitmap */
   static HBRUSH hBrush; /* store the brush handle */
   static HFONT holdf, hnewf1, hnewf2;
   /* store the font handles */
   static char fname[40] = "Default"; /* name of font */
```

```
switch( uMsg )
{
                       case WM_CREATE:
                       /* get screen coordinates */
      maxX = GetSystemMetrics(SM_CXSCREEN);
      maxY = GetSystemMetrics(SM_CYSCREEN);

      /* make a compatible memory image device */
      hDC = GetDC(hWnd);
      memdc = CreateCompatibleDC(hDC);
      hBit = CreateCompatibleBitmap(hDC, maxX, maxY);
      SelectObject(memdc, hBit);
      hBrush = (HBRUSH)GetStockObject(WHITE_BRUSH);
      SelectObject(memdc, hBrush);
      PatBlt(memdc, 0, 0, maxX, maxY, PATCOPY);

         /* create a new font */
      hnewf1 = CreateFont(14, 0, 0, 0, FW_NORMAL,
                       0, 0, 0, ANSI_CHARSET,
                       OUT_DEFAULT_PRECIS,
                       CLIP_DEFAULT_PRECIS,
                       DEFAULT_QUALITY,
                       DEFAULT_PITCH | FF_DONTCARE,
                       "Courier New");
      hnewf2 = CreateFont(20, 0, 0, 0, FW_SEMIBOLD,
                       0, 0, 0, ANSI_CHARSET,
                       OUT_DEFAULT_PRECIS,
                       CLIP_DEFAULT_PRECIS,
                       DEFAULT_QUALITY,
                       DEFAULT_PITCH | FF_DONTCARE,
                       "Century Gothic");
      ReleaseDC(hWnd, hDC);
      break;
   case WM_COMMAND :
        switch( LOWORD( wParam ) )
        {
              case IDM_TEST :
        /* set text color to black and mode to transparent */
                    SetTextColor(memdc, RGB(0, 0, 0));
                    SetBkMode(memdc, TRANSPARENT);
```

```
          /* get text metrics */
          GetTextMetrics(memdc, &tm);

          sprintf(str, "The font is %ld pixels
                    high.", tm.tmHeight);
          TextOut(memdc, X, Y, str, strlen(str));
          /* output string */
          Y = Y + tm.tmHeight + tm.tmExternalLeading;
          /* next line */

          strcpy(str, "This is on the next line. ");
          TextOut(memdc, X, Y, str, strlen(str));
          /* output string */

          /* compute length of a string */
          GetTextExtentPoint32(memdc, str,
             strlen(str), &size);
          sprintf(str, "Previous string is %ld units
             long", size.cx);
          X = size.cx;
          /* advance to end of previous string */
          TextOut(memdc, X, Y, str, strlen(str));
          Y = Y + tm.tmHeight + tm.tmExternalLeading;
          /* next line */
          X = 0; /* reset X */

          sprintf(str, "Screen dimensions: %d %d",
             maxX, maxY);
          TextOut(memdc, X, Y, str, strlen(str));
          Y = Y + tm.tmHeight + tm.tmExternalLeading;
          /* next line */
          InvalidateRect(hWnd, NULL, 1);
          break;

     case IDM_CHANGE_FONT:
          switch(fontswitch) {
               case 0: /* switch to new font1 */
                    holdf = (HFONT)SelectObject
                          (memdc, hnewf1);
                    fontswitch = 1;
                    strcpy(fname, "Courier New");
                    break;
```

```
                              case 1: /* switch to new font2 */
                                    SelectObject(memdc, hnewf2);
                                    fontswitch = 2;
                                    strcpy(fname, "Century
                                           Gothic");
                                    break;

                             default: /* switch to old font */
                                    SelectObject(memdc, holdf);
                                    fontswitch = 0;
                                    strcpy(fname, "Default");
                                                   }
                                                   break;
                case IDM_ABOUT :
                       DialogBox( hInst, "AboutBox", hWnd,
                              (DLGPROC)About );
                       break;

                case IDM_EXIT :
                       DestroyWindow( hWnd );
                       break;
             }
             break;
    case WM_PAINT: /* process a repaint request */
      hDC = BeginPaint(hWnd, &paintstruct); /* get DC */

      /* now, copy memory image onto screen */
      BitBlt(hDC, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
      EndPaint(hWnd, &paintstruct); /* release DC */
      break;
      case WM_DESTROY :
              DeleteDC(memdc); /* delete the memory device */
              PostQuitMessage(0);
              break;

      default :
            return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
   }

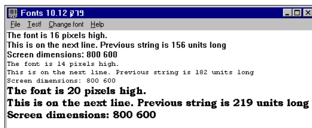
   return( 0L );
}
```

```
LRESULT CALLBACK About( HWND hDlg,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
   switch (message)
   {
       case WM_INITDIALOG:
               return (TRUE);

       case WM_COMMAND:
               if (   LOWORD(wParam) == IDOK
                   || LOWORD(wParam) == IDCANCEL)
               {
                       EndDialog(hDlg, TRUE);
                       return (TRUE);
               }
               break;
   }

   return (FALSE);
}
```

פלט לדוגמה מופיע בתרשים 10.3.



תרשים 10.3: פלט לדוגמה מהתוכנית המפיקה גופנים אישיים

10.13 הפונקציה EnumFontFamilies

Windows מאגדת גופנים למשפחות, בהתבסס על סדרת מאפיינים משותפים לכל גופן במשפחה. באפשרות התוכניות להשתמש בפונקציה EnumFontFamilies כדי לזהות במשפחת גופנים מוגדרת את הגופנים הזמינים בהתקן מוגדר. בדרך כלל משתמשים בפונקציה EnumFontFamilies כדי לקבוע באיזה גופנים באפשרותך להשתמש בהתקן מוגדר, וגם כדי להשיג מצביע למבנה LOGFONT שהתוכנית יכולה להשתמש בו יחד עם הפונקציה EnumFontFamilies כדי ליצור את הגופן עבור ההתקן המוגדר. את הפונקציה EnumFontFamilies כותבים כמו בהגדרה שלהלן:

```
int EnumFontFamilies(
  HDC hdc,              // handle to device context
  LPCTSTR lpszFamily,   // pointer to family-name string
  FONTENUMPROC lpEnumFontFamProc,
                        // pointer to callback function
  LPARAM lParam         // address of application-supplied data
);
```

טבלה 10.1 מפרטת את הפרמטרים שהפונקציה EnumFontFamilies מקבלת.

טבלה 10.1: הפרמטרים שהפונקציה **EnumFontFamilies** מקבלת.

פרמטר	תיאור
hdc	מזהה את הקשר ההתקן.
lpszFamily	מצביע למחרוזת המסתיימת ב-NULL, אשר מגדירה את שם המשפחה של הגופן המבוקש. אם lpszFamily שווה ל-NULL, הפונקציה EnumFontFamilies בוחרת באקראי וממספרת גופן אחד בכל משפחה זמינה.
lpEnumFontFamProc	מגדיר את כתובת המופע של פרוצדורת המשוב שמוגדרת על ידי היישום. למידע אודות **פונקציית המשוב** (Callback Function), התבונן בפונקציה EnumFontFamProc.
lParam	מצביע לנתונים המסופקים על ידי היישום. הפונקציה מעבירה את הנתונים לפונקציית המשוב יחד עם המידע אודות הגופן.

כאשר הפונקציה EnumFontFamilies מצליחה, הערך המוחזר הוא הערך האחרון שהחזירה פונקציית המשוב. זאת אומרת שהמשמעות נקבעת על פי המימוש המסוים שלו. הפונקציה EnumFontFamilies מקבלת את שמות הסגנונות שקשורים עם גופן TrueType. עם EnumFontFamilies, באפשרותך לקבל מידע על סגנונות גופן לא רגילים (לדוגמה, מיתאר, outline). הפונקציה EnumFontFamilies מקבלת מידע אודות כל גופן בעל שם של צורת גופן (typeface) שמוגדר על ידי lpszFamily, ומוסרת אותו לפונקציה

אשר lpEnumFontFamProc מצביעה עליה. פונקציית המשוב המוגדרת על ידי היישום יכולה לעבד את מידע הגופן כנדרש. הספירה נמשכת, עד שאין גופנים נוספים, או עד שפונקציית המשוב מחזירה אפס. את פונקציית המשוב כותבים כמו בהגדרה שלהלן:

```
int CALLBACK EnumFontFamProc(
  ENUMLOGFONT* lpelf,    // pointer to an ENUMLOGFONT structure
  NEWTEXTMETRIC* lpntm,  // pointer to NEWTEXTMETRIC structure.
  int nFontType,         // The font type
  LPARAM lParam          // Application-defined data
);
```

כמו שאפשר לראות, פונקציית המשוב מקבלת ארבעה פרמטרים. הראשון הוא מצביע למבנה מסוג ENUMLOGFONT. המבנה ENUMLOGFONT מגדיר את מאפייני הגופן, השם המלא של הגופן וסגנון הגופן.

המבנה ENUMLOGFONT מוגדר ב- Win32 API כפי שרואים להלן:

```
typedef struct tagENUMLOGFONT {
  LOGFONT   elfLogFont;
  BCHAR     elfFullName[LF_FULLFACESIZE];
  BCHAR     elfStyle[LF_FACESIZE];
} ENUMLOGFONT;
```

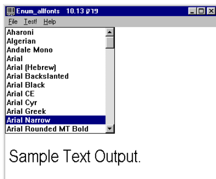
טבלה 10.2 מפרטת את איברי המבנה ENUMLOGFONT.

טבלה 10.2: איברי המבנה ENUMLOGFONT.

פרמטר	תיאור
elfLogFont	מתייחס למבנה מסוג LOGFONT אשר מגדיר את מאפייני הגופן.
elfFullName	מגדיר שם ייחודי לגופן. לדוגמה, "ABCD Font Company TrueType Bold Italic Sans Serif".
elfStyle	מגדיר את סגנון הגופן, כמו לדוגמה, "Bold Italic". המבנה TEXTMETRIC מכיל מידע בסיסי אודות גופן פיזי. כל הגדלים נתונים ביחידות לוגיות; כלומר, הם תלויים במצב המיפוי הנוכחי של תצוגת ההקשר.

שלושת הפרמטרים הנותרים מגדירים מידע TrueType ייחודי ואת סוג הגופן (מתוך DEVICE_FONTTYPE, RASTER_FONTTYPE, או TRUETYPE_FONTTYPE, או צירוף כלשהו של שלושת הערכים).

כדי להבין יותר טוב את פעולת הפונקציה EnumFontFamilies, התבונן בתוכנית **Enum_AllFonts**, שבתקליטור המצורף לספר זה (בתיקיה Chap10\Enum_ AllFonts). התוכנית משתמשת בפונקציה EnumFontFamilies כדי למלא תיבת רשימה (List Box) הכוללת את שמות הגופנים הזמינים מסוג TrueType. כאשר המשתמש בוחר באפשרות Test! מהתפריט, התוכנית מציגה דוגמה למחרוזת טקסט בפורמט הגופן שנבחר כרגע. הקוד המעשי של התוכנית **Enum_AllFonts** נמצא בפונקציות WndProc, EnumFontProc, ו-FindFontProc. הפלט לדוגמה מופיע בתרשים 10.4.



תרשים 10.4: פלט לדוגמה מהתוכנית **Enum_AllFonts**.

10.14 הצגת גופנים רבים עם CreateFontIndirect

למדת שתוכניות יכולות להשתמש בפונקציה CreateFont כדי ליצור גופנים. עם זאת, מספר הפרמטרים בקריאה אחת ל-CreateFont מספיק בכדי להפוך את השימוש ב-CreateFont למתיש. חלופה טובה יותר היא הפונקציה CreateFontIndirect, אשר יוצרת גופן לוגי שיש לו את כל המאפיינים שמוגדרים במבנה מסוים. כתוצאה, באפשרותך לבחור את הגופן כגופן הנוכחי בהקשר התקן כלשהו, כמו בדוגמה זו:

```
HFONT CreateFontIndirect(CONST LOGFONT *lplf);
```

הפרמטר lplf מצביע אל מבנה מסוג LOGFONT אשר מגדיר מאפיינים של גופן לוגי. המבנה LOGFONT מגדיר את המאפיינים של הגופן, כמו שרואים להלן:

```
typedef struct tagLOGFONT {
                        LONG lfHeight;
                        LONG lfWidth;
                        LONG lfEscapement;
                        LONG lfOrientation;
                        LONG lfWeight;
                        BYTE lfItalic;
                        BYTE lfUnderline;
                        BYTE lfStrikeOut;
                        BYTE lfCharSet;
                        BYTE lfOutPrecision;
                        BYTE lfClipPrecision;
                        BYTE lfQuality;
                        BYTE lfPitchAndFamily;
                        TCHAR lfFaceName[LF_FACESIZSE];
} LOGFONT;
```

כאשר מתבוננים מקרוב במבנה LOGFONT, אפשר לראות שאיבריו מתאימים לפרמטרים של הפונקציה CreateFont. למעשה, האיברים מקבלים ערכים כמו הפרמטרים של הפונקציה CreateFont.

אם הפונקציה CreateFontIndirect מצליחה, הערך המוחזר הוא ידית לגופן לוגי; אם הפונקציה CreateFontIndirect נכשלת, הערך המוחזר הוא NULL. הפונקציה CreateFontIndirect יוצרת גופן לוגי עם המאפיינים שמוגדרים במבנה LOGFONT. כאשר בוחרים את הגופן עם הפונקציה SelectObject, ממפה הגופנים של ממשק ההתקן הגרפי מנסה להתאים את הגופן הלוגי עם גופן פיסי קיים. אם ממשק ההתקן הגרפי אינו יכול למצוא התאמה מדויקת, הוא מספק חלופה אשר מאפייניה תואמים את המאפיינים הדרושים ככל האפשר. כאשר אין צורך יותר בגופן, צריך לקרוא לפונקציה DeleteObject כדי למחוק אותו.

התוכנית **Enum_AllFonts** שהוצגה בסעיף הקודם משתמשת בפונקציה CreateFontIndirect. במקום הקריאה ל-CreateFont עם ארבע עשר הפרמטרים, התוכנית קוראת ל-CreateFontIndirect עם פרמטר אחד בלבד, כפי שמוצג להלן:

```
hFont = CreateFontIndirect(&lf);
```

זכור, התמיכה של חלונות לגופנים ולטקסט רחבה למדי. בוודאי תרצה לחקור תחום זה בעצמך. בפרק הבא נמשיך ונבדוק את האפשרויות למשלוח פלט לחלון באמצעות עבודה עם גרפיקה.

פרק **11**

עבודה בגרפיקה

מערכת חלונות 9x כוללת אוסף עשיר וגמיש של פונקציות גרפיות, לנוחותו של המתכנת. אין זה מפתיע, שהרי זוהי מערכת הפעלה גרפית. אך תופתע אולי מהמידה שבה שלובה הגרפיקה במערכת התצוגה החלונאית. למעשה, חלק גדול ממה שלמדת אודות טקסט בפרק הקודם, יפה גם לגרפיקה. לדוגמה, המברשת המשמשת לצביעת חלון, משמשת גם למילוי של עצם. בפרק זה נדון בכמה מהפונקציות הגרפיות של חלונות, ונדגים כיצד הן פועלות.

הפרק גם בוחן מספר מאפיינים הקובעים בדיוק כיצד ימופה פלט נתון על החלון שאליו הוא מיועד. נדון בנושאים כגון קביעת מצב המיפוי הנוכחי, כיצד לשנות את הקואורדינטות הלוגיות הקשורות בחלון נתון, וכיצד להגדיר את שטח התצוגה (viewport). לגורמים אלה השפעה מכרעת על אופן הצגת הגרפיקה והטקסט על המסך.

זכור שהדיון בגרפיקה ובנושאים הקשורים לגרפיקה בפרק זה אינו אלא נגיעה בפני השטח. מערכת הגרפיקה של חלונות 9x היא בעלת עוצמה, ויש להניח שתרצה להמשיך ולחקור אותה בעצמך.

11.1 מערכת הקואורדינטות לגרפיקה

מערכת הקואורדינטות לגרפיקה היא זו המשמשת את הפונקציות המבוססות על טקסט (הנדונות בפרק 10). פירוש הדבר שלפי ברירת המחדל, הפינה השמאלית-העליונה ממוקמת בנקודה 0,0 ; והיחידות הלוגיות שקולות כנגד פיקסלים. זכור שמערכת הקואורדינטות והמיפוי של יחידות לוגיות למתכונת של פיקסלים נתונים בשליטתך וניתן לשנותם (בהמשך הפרק יתברר לך כיצד).

חלונות 9x שומרת כל הזמן ערך של **מיקום נוכחי** (Current Position), המשמש פונקציות גרפיות מסוימות, שגם מעדכנות אותו. כאשר התוכנית שלך מתחילה לרוץ, מתאפס המיקום הנוכחי ומוצב על 0,0. זכור שהנקודה המייצגת את המיקום הנוכחי היא בלתי נראית. כלומר, אין שום "סמן" גרפיקה על המסך. המיקום הנוכחי הוא המקום הבא בחלון, שבו יתחילו לפעול פונקציות גרפיות מסוימות.

11.2 עטים ומברשות

מערכת הגרפיקה החלונאית מבוססת על שני עצמים חשובים: עטים ומברשות. כבר עסקנו במברשות בפרק 10. כל המידע הזה חל גם על הפונקציות הגרפיות המתוארות בפרק. לפי ברירת המחדל, צורות גרפיות סגורות, כגון מלבנים ואליפסות, מתמלאים בעזרת המברשת הנוכחית. **עטים** (Pens) הם משאב המצייר קווים וקשתות לפי הוראות מפונקציות גרפיות מסוימות. לפי ברירת המחדל, צבע העט שחור, והוא מצייר קו בעובי פיקסל אחד. בידך האפשרות לשנות את התכונות האלו.

עד כה עבדת רק עם עצמים ממלאי המערכת. בפרק זה תלמד כיצד ליצור מברשות ועטים מותאמים אישית. זכור דבר חשוב אחד לגבי עצמים אישיים שהתאמת לעצמך: חובה למחוק אותם לפני סיום התוכנית באמצעות הפונקציה ()DeleteObject.

11.3 הגדרה של פיקסלים

תוכל לקבוע את צבעו של כל פיקסל נתון באמצעות הפונקציה ()SetPixel של API, שהגדרתה הכללית היא:

```
COLORREF SetPixel(HDC hdc, int X, int Y, COLORREF color);
```

במקרה זה, hdc היא הידית להקשר ההתקן הרצוי. הקואורדינטות של הפיקסל שאת הגדרתו מבקשים לקבוע נתונות על ידי X ו-Y, והפרמטר color מכיל את הגדרת הצבע (דיון בסוג הנתונים COLORREF מופיע בפרק 10). הפונקציה מחזירה את הצבע המקורי של הפיקסל, וערך 1- במקרה של שגיאה, או במקרה שהמיקום שצוין נמצא מחוץ לגבולות החלון.

11.4 שרטוט קווים

כדי לשרטט קו, השתמש בפונקציה ()LineTo, אשר מציירת קו בעזרת העט הנוכחי. לפניך הגדרתה הכללית:

```
BOOL LineTo(HDC hdc, int X, int Y);
```

hdc מכיל את הידית להקשר ההתקן שבתוכו יש לצייר את הקו. הקו ישורטט מהמיקום הגרפי הנוכחי, אל הנקודה הנתונה על ידי הקואורדינטות X ו-Y. לאחר מכן יעודכן המיקום הנוכחי ויקבל את הערכים X,Y. הפונקציה מחזירה ערך לא-אפס אם סיימה בהצלחה (כלומר, הקו צויר), וערך אפס אם לא.

יש מתכנתים שמופתעים מכך שהפונקציה ()LineTo משתמשת במיקום הנוכחי כנקודת יציאה, ולאחר מכן מגדירה את המיקום הנוכחי לפי נקודת הסיום של הקו ששורטט (במקום להותירה בלא שינוי). יש לכך סיבה טובה. פעמים רבות, כאשר מציגים קווים, קורה שקו אחד מתחיל בסופו של הקו הקודם. במקרה כזה, הפונקציה ()LineTo פועלת באורח יעיל ביותר כי אינה צריכה להעביר זוג קואורדינטות נוסף. כשאינך מעוניין במתכונת כזו, עליך לקבוע את המיקום הנוכחי בכל נקודה הרצויה לך באמצעות הפונקציה ()MoveToEx שמייד נעסוק בה, עוד לפני הקריאה לפונקציה ()LineTo.

11.5 קביעת המיקום הנוכחי

כדי לקבוע מיקום נוכחי, השתמש בפונקציה MoveToEx()‎, שהגדרתה הכללית היא:

```
BOOL MoveToEx(HDC hdc, int X, int Y, lPPOINT lpCoord);
```

הפרמטר hdc מכיל את הידית להקשר ההתקן. הקואורדינטות של המיקום הנוכחי החדש נתונות על ידי X,Y. ‏lpCoord הוא מצביע אל מבנה POINT, המחזיר את המיקום הנוכחי הקודם. לפניך הגדרת מבנה POINT:

```
typedef struct tagPOINT {
  LONG x;
  LONG y;
} POINT;
```

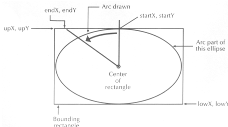
אם תקצה לפרמטר lpCoord ערך NULL, הפונקציה MoveToEX()‎ לא תחזיר את המיקום הנוכחי הקודם.

הפונקציה MoveToEx()‎ מחזירה ערך לא-אפס אם סיימה בהצלחה, וערך אפס אם לא.

11.6 ציור קשתות

תוכל לצייר **קשת אליפטית** (Eliptical Arc), מקטע של אליפסה) בצבע העט הנוכחי באמצעות הפונקציה Arc()‎, שתבניתה הכללית היא:

```
BOOL Arc(HDC hdc, int upX, int upY, int lowX, int lowY,
                  int startX, int startY, int endx, int endY);
```



תרשים 11.1: אופן הפעולה של הפונקציה Arc()‎

בדוגמה זו, hdc היא הידית להקשר ההתקן שבו תצויר הקשת. הגדרת הקשת נעשית באמצעות שני עצמים. הראשון, הקשת עצמה, הוא חלק של אליפסה הכלואה במלבן (Bounding Rectangle) שפינתו השמאלית-העליונה נמצאת בנקודה upX,upY, ופינתו

הימנית-התחתונה בנקודה lowX,lowY. אותו חלק של האליפסה המצויר בפועל (הקשת), מתחיל בנקודה שבה מצטלבים הקו העובר דרך מרכז המלבן והנקודה הנתונה על ידי startX,startY. הוא מסתיים בנקודה שבה מצטלבים הקו העובר דרך מרכז המלבן ודרך הנקודה הנתונה על ידי endX,endY. הקשת מצוירת בניגוד לכיוון השעון, מהנקודה startX,startY. בתרשים 11.1 מודגמת פעולת הפונקציה ()Arc.

הפונקציה ()Arc מחזירה ערך לא-אפס אם סיימה בהצלחה, וערך אפס אם לא.

11.7 כיצד להציג מלבנים

באפשרותך לחולל תצוגת **מלבן** (Rectangle), שיצויר בעט הנוכחי, באמצעות הפונקציה ()Rectangle, שהגדרתה מובאת כאן:

```
BOOL Rectangle(HDC hdc, int upX, int upY, int lowX, int lowY);
```

hdc היא הידית להקשר ההתקן. הפינה שמאלית-העליונה של המלבן נתונה על ידי upX,upY, והפינה הימנית-התחתונה נתונה על ידי lowX,lowY. הפונקציה מחזירה ערך לא-אפס אם סיימה בהצלחה, ואפס במקרה של שגיאה. המלבן יתמלא באופן אוטומטי, על ידי המברשת הנוכחית.

תוכל להציג **מלבן מעוגל** (Rounded Rectangle), מלבן שפינותיו מעוגלות במקצת, באמצעות הפונקציה ()RoundRect, שהגדרתה הכללית מובאת להלן:

```
BOOL RoundRect(HDC hdc, int upX, int upY, int lowX,
                              int lowY, int curveX, int curveY);
```

חמשת הפרמטרים הראשונים זהים לאלה המופיעים בפונקציה ()Rectangle. צורת עיגול הפינות נקבעת על ידי ערכי הפרמטרים curveX ו-curveY, המגדירים את רוחבה וגובהה של האליפסה המתווה את הקשת. הפונקציה מחזירה ערך לא-אפס אם סיימה בהצלחה, וערך אפס במקרה של שגיאה. המלבן המעוגל מתמלא באופן אוטומטי על ידי המברשת הנוכחית.

11.8 כיצד לצייר אליפסות ופרוסות עוגה

כדי לצייר אליפסה (Ellipse) או עיגול בעזרת העט הנוכחי, עליך להשתמש בפונקציה ()Ellipse, שהגדרתה הכללית היא:

```
BOOL Ellipse(HDC hdc, int upX, int upY, int lowX, int lowY);
```

בדוגמה, hdc היא הידית של הקשר ההתקן שבו תצויר האליפסה. כל אליפסה מוגדרת באמצעות המלבן התוחם אותה. הפינה השמאלית-העליונה של המלבן נתונה על ידי upX,upY, והפינה הימנית-התחתונה נתונה על ידי lowX,lowY. כדי לצייר עיגול, הגדר ריבוע במקום מלבן.

הפונקציה מחזירה ערך לא-אפס אם סיימה בהצלחה, וערך אפס אם לא. האליפסה מתמלאת על ידי המברשת הנוכחית.

פרוסת העוגה (Pie Slice) היא צורה קרובה לאליפסה. פרוסת עוגה היא עצם המורכב מקשת ומשני קווים אל מרכז האליפסה, אחד מכל קצה של הקשת. כדי לצייר פרוסת עוגה, השתמש בפונקציה ()Pie, שהגדרת הכללית מובאת להלן:

```
BOOL Pie(HDC hdc, int upX, int upY, int lowX, int lowY,
         int startX, int startY, int endX, int endY);
```

במקרה זה, hdc היא הידית להקשר ההתקן שבו תצויר פרוסת העוגה. הקשת של הפרוסה מוגדרת באמצעות שני עצמים. הקשת היא חלק מאליפסה, הכלואה במלבן שפינתו השמאלית-העליונה ממוקמת בנקודה upX,upY, ופינתו הימנית-התחתונה בנקודה lowX,lowY. אותו חלק של האליפסה המצויר בפועל (הקשת של הפרוסה), מתחילה בהצטלבות של קו היוצא ממרכז המלבן ועובר דרך הנקודה הנתונה על ידי startX,startY, ומסתיימת בהצטלבות של קו היוצא ממרכז המלבן ועובר דרך הנקודה endX,endY. הפרוסה מצוירת בעט הנוכחי ומתמלאת על ידי המברשת הנוכחית.

הפונקציה ()Pie מחזירה ערך לא-אפס אם סיימה בהצלחה, וערך אפס במקרה של שגיאה.

11.9 כיצד לעבוד עם עטים

עצמים גרפיים מצוירים בעזרת העט הנוכחי. במלאי המערכת קיימים שלושה עטים: black, white ו-null. ניתן להשיג ידית לכל אחד מהם באמצעות הפונקציה ()GetStockObject, שנדונה בשלב מוקדם יותר בספר. פקודות המאקרו עבור עטי המערכת הן BLACK_PEN, WHITE_PEN ו-NULL_PEN, בהתאמה. ידיות לעטים הן מסוג HPEN.

למעשה, **עטי המערכת** (Stock Pens) מוגבלים למדי ובדרך כלל תגלה שנוח לך להגדיר עטים משלך עבור היישום שאתה כותב. הדבר מתבצע בעזרת הפונקציה ()CreatePen, שהגדרתה הכללית היא:

```
HPEN CreatePen(int style, int width, COLORREF color);
```

הפרמטר style קובע את סוג העט שייווצר, וחייב להכיל אחד מהערכים הבאים:

שם המאקרו	סגנון העט
PS_DASH	מקווקוו
PS_DASHDOT	קו-נקודה
PS_DASHDOTDASH	קו-נקודה-קו
PS_DOT	מנוקד
PS_INSIDEFRAME	עט מלא בתוך גבולות של אזור תחום
PS_NULL	אין
PS_SOLID	קו רצוף

הסגנונות המנוקדים ו/או המקווקווים חלים על עטים בעובי של יחידה אחת בלבד. העט PS_INSIDEFRAME הוא עט מלא, שיהיה כולו בתוך גבולותיו של העצם שיצויר, גם כאשר עובי העט עולה על יחידה אחת. לדוגמה, אם עט בסגנון PS_INSIDEFRAME ובעובי העולה על יחידה אחת משמש לציור מלבן, אזי צידו החיצוני של הקו יהיה בתוך גבולות המלבן (כאשר משתמשים בעט רחב מסגנון אחר, ייתכן שהקו יחרוג בחלקו מגבולות העצם).

עובי העט מוגדר באמצעות הפרמטר width, שהערך שלו הוא ביחידות לוגיות. הפרמטר color, שהוא ערך של COLORREF (שנדון בפרק 10), קובע את צבע העט. בדוגמאות של פרק זה יוגדרו כל הצבעים בעזרת ערכי RGB.

לאחר שנוצר העט, הוא מוצב בהקשר התקן באמצעות הפונקציה ()SelectObject. לדוגמה, קטע הקוד הבא יוצר עט אדום ולאחר מכן בוחר בו להיות העט הנוכחי:

```
HPEN hRedpen;
hRedpen = CreatePen(PS_SOLID, 1, rgb(255, 0, 0);
SelectObject(dc, hRedpen);
```

זכור, לפני סיום התוכנית עליך **למחוק** כל עט אישי שיצרת באמצעות הפונקציה ()DeleteObject.

11.10 כיצד ליצור מברשות אישיות

מברשות המותאמות אישית לצרכיך (Custom Brushes) נוצרות בצורה דומה לעטים אישיים. קיימים מספר סגנונות של מברשות, הנפוצה היא solid brush. מברשת מלאה נוצרת בעזרת הפונקציה CreateSolidBrush, שהגדרתה הכללית היא:

```
HBRUSH CreateSolidBrush(COLORREF color);
```

צבע המברשת מוגדר באמצעות ערך הפרמטר color, והפונקציה מחזירה ידית לאותה מברשת.

לאחר שנוצרה המברשת האישית, היא מוצבת בהקשר ההתקן באמצעות הפונקציה ()SelectObject. לדוגמה, קטע הקוד הבא יוצר מברשת ירוקה, ולאחר מכן בוחר בה להיות המברשת הנוכחית.

```
HBRUSH hGreenbrush
hGreenbrush = CreateSolidBrush(RGB(0, 255, 0));
SelectObject(dc, hGreenbrush);
```

בדומה לעטים אישיים, **חובה** למחוק גם מברשות אישיות לפני סיום התוכנית.

סוגים נוספים של מברשות, שבוודאי תרצה לחקור בעצמך, הם מברשות עם קווקווים או עם דוגמאות, הנוצרות באמצעות הפונקציות ()CreateHatchBrush וב-()CreatePatternBrush, בהתאמה.

11.11 מחיקת עצמים אישיים

חובה למחוק **עצמים אישיים** (Custom Objects) לפני סיום התוכנית. פעולה זו מתבצעת באמצעות הפונקציה ()DeleteObject. זכור, **אין** למחוק עצם המוצב באותו זמן בהקשר התקן כלשהו.

11.12 הדגמת עבודה בגרפיקה

התוכנית **Gui11** שלפניך מדגימה את הפונקציות הגרפיות השונות שתוארו לעיל. תוכנית זו משתמשת בטכניקת החלון הווירטואלי שעסקנו בה בפרק 10. היא מכוונת את הפלט להקשר התקן המצוי בזיכרון. לאחר מכן מועתק תוכן החלון אל החלון הפיסי בכל פעם שמתקבלת הודעת WM_PAINT (זכור, גישה זו לפלט מאפשרת עדכון אוטומטי של תוכן החלון כאשר מתקבלת הודעת WM_PAINT) התוכנית בשלמותה נמצאת בתקליטור Chap11\Gui11.

```
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include "Gui11.h"
#include "resource.h"

/* Demonstrate Gui functions . */

#if defined (WIN32)
                        #define IS_WIN32 TRUE
#else
                        #define IS_WIN32 FALSE
#endif

#define IS_NT       IS_WIN32 && (BOOL)(GetVersion() < 0x80000000)
#define IS_WIN32S   IS_WIN32 && (BOOL)(!(IS_NT) &&
                    (LOBYTE(LOWORD(GetVersion()))<4))
#define IS_WIN95    (BOOL)(!(IS_NT) && !(IS_WIN32S)) && IS_WIN32

HINSTANCE hInst;   // current instance
BOOL RegisterWin95( CONST WNDCLASS* lpwc );

LPCTSTR lpszAppName  = "MyApp";
LPCTSTR lpszTitle    = " Gui11  פרק 11 ";
char str[255]; /* holds output strings */
```

```
int X=0, Y=0; /* current output location */
int maxX, maxY; /* screen dimensions */

int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                 hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
{
   MSG      msg;
   HWND     hWnd;
   WNDCLASS wc;

   // Register the main application window class.
   //...........................................
   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = (WNDPROC)WndProc; /* window function */
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInstance;  /* handle to this instance */
   wc.hIcon         = LoadIcon( hInstance, lpszAppName );
                      /* icon style */
   wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
                      /* cursor style */
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName; /* window class name */

   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );

    hInst = hInstance;

   // Create the main application window.
   //...................................
   hWnd = CreateWindow( lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
```

```
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                      );

   if ( !hWnd )
      return( FALSE );

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );

   while( GetMessage( &msg, NULL, 0, 0) )
   {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }

   return( msg.wParam );

}

BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style         = lpwc->style;
   wcex.lpfnWndProc   = lpwc->lpfnWndProc;
   wcex.cbClsExtra    = lpwc->cbClsExtra;
   wcex.cbWndExtra    = lpwc->cbWndExtra;
   wcex.hInstance     = lpwc->hInstance;
   wcex.hIcon         = lpwc->hIcon;
   wcex.hCursor       = lpwc->hCursor;
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;
```

```
    // Added elements for Windows 95.
    //..............................
    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");

    return RegisterClassEx( &wcex );
}

/* This function is called by Windows 95 and is passed
   messages from the message queue.*/
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam )
{
    HDC hDC;
    PAINTSTRUCT paintstruct;
    static HPEN hRedpen, hGreenpen, hBluepen, hYellowpen, hOldpen;

    static HDC memdc; /* store the virtual device handle */
    static HBITMAP hBit; /* store the virtual bitmap */
    static HBRUSH hBrush, hOldbrush; /* store the brush handle */

    switch( uMsg )
    {
          case WM_CREATE:
          maxX = GetSystemMetrics(SM_CXSCREEN);
          maxY = GetSystemMetrics(SM_CYSCREEN);

          /* make a compatible memory image device */
          hDC = GetDC(hWnd);
          memdc = CreateCompatibleDC(hDC);
          hBit = CreateCompatibleBitmap(hDC, maxX, maxY);
          SelectObject(memdc, hBit);
          hBrush = (HBRUSH)GetStockObject(WHITE_BRUSH);
          SelectObject(memdc, hBrush);
          PatBlt(memdc, 0, 0, maxX, maxY, PATCOPY);

          hRedpen = CreatePen(PS_SOLID, 1, RGB(255,0,0));
          hGreenpen = CreatePen(PS_SOLID, 2, RGB(0,255,0));
          hBluepen = CreatePen(PS_SOLID, 3, RGB(0,0,255));
          hYellowpen = CreatePen(PS_SOLID, 4, RGB(255, 255, 0));
```

```
        /* save default pen */
        hOldpen = (HPEN)SelectObject(memdc, hRedpen);
        SelectObject(memdc, hOldpen);

        ReleaseDC(hWnd, hDC);
        break;
    case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                    case IDM_LINES:
                    /* set 2 pixels */
                    SetPixel(memdc, 40, 14, RGB(0, 0, 0));
                    SetPixel(memdc, 40, 15, RGB(0, 0, 0));

                    LineTo(memdc, 100, 50);
                    MoveToEx(memdc, 100, 50, NULL);

                    /* change to green pen */
                    hOldpen = (HPEN)SelectObject(memdc,
                                hGreenpen);
                    LineTo(memdc, 200, 100);

                    /* change to yellow pen */
                    SelectObject(memdc, hYellowpen);
                    LineTo(memdc, 0, 200);

                    /* change to blue pen */
                    SelectObject(memdc, hBluepen);
                    LineTo(memdc, 200, 200);

                    /* change to red pen */
                    SelectObject(memdc, hRedpen);
                    LineTo(memdc, 0, 0);

                    /* return to default pen */
                    SelectObject(memdc, hOldpen);

                    Arc(memdc, 0, 0, 300, 300, 0, 50, 200, 50);

                   /* show intersecting lines that define arc */
                    MoveToEx(memdc, 150, 150, NULL);
                    LineTo(memdc, 0, 50);
                    MoveToEx(memdc, 150, 150, NULL);
                    LineTo(memdc, 200, 50);
```

```
            InvalidateRect(hWnd, NULL, 1);
            break;
         case IDM_RECTANGLES:
            /* display, but don't fill */
            hOldbrush = (HBRUSH)SelectObject(memdc,
                          GetStockObject(HOLLOW_BRUSH));

            /* draw some rectangles */
            Rectangle(memdc, 50, 50, 300, 300);
            RoundRect(memdc, 125, 125, 220, 240, 15,
                        13);

            /* use a red pen */
            SelectObject(memdc, hRedpen);
            Rectangle(memdc, 100, 100, 200, 200);
            SelectObject(memdc, hOldpen);
            /* return to default pen */

            /* restore default brush */
            SelectObject(memdc, hOldbrush);

            InvalidateRect(hWnd, NULL, 1);
            break;
         case IDM_ELLIPSES:
            /* make blue brush */
            hBrush = (HBRUSH)CreateSolidBrush(RGB(0, 0,
                        255));
            hOldbrush = (HBRUSH)SelectObject(memdc,
                          hBrush);

            /* fill these ellipses with blue */
            Ellipse(memdc, 50, 200, 100, 280);
            Ellipse(memdc, 75, 25, 280, 100);

            /* use a red pen and fill with green */
            SelectObject(memdc, hRedpen);
            DeleteObject(hBrush); /* delete brush */
```

```
            /* create green brush */
            hBrush = CreateSolidBrush(RGB(0, 255, 0));
            SelectObject(memdc, hBrush);
            /* select green brush */
            Ellipse(memdc, 100, 100, 200, 200);

            /* draw a pie slice */
            Pie(memdc, 200, 200, 340, 340, 225, 200,
                200, 250);

            SelectObject(memdc, hOldpen);
            /* return to default pen */
            SelectObject(memdc, hOldbrush);
            /* select default brush */
            DeleteObject(hBrush);
            /* delete green brush */
            InvalidateRect(hWnd, NULL, 1);
            break;

        case IDM_CLEAR:
           /* reset current position to 0,0 */
           MoveToEx(memdc, 0, 0, NULL);

           /* erase by repainting background */
           PatBlt(memdc, 0, 0, maxX, maxY, PATCOPY);
           InvalidateRect(hWnd, NULL, 1);
               break;
        case IDM_ABOUT :
                DialogBox( hInst, "AboutBox", hWnd,
               (DLGPROC)About );
                break;

        case IDM_EXIT :
                DestroyWindow( hWnd );
                break;
    }
    break;
```

```
      case WM_PAINT: /* process a repaint request */
           hDC = BeginPaint(hWnd, &paintstruct); /* get DC */

           /* now, copy memory image onto screen */
           BitBlt(hDC, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
           EndPaint(hWnd, &paintstruct); /* release DC */
           break;
      case WM_DESTROY :
                               DeleteDC(memdc);
                              /* delete the memory device */
           PostQuitMessage(0);
           break;
      default :
          return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
   }

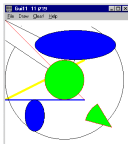
   return( 0L );
}

LRESULT CALLBACK About( HWND hDlg,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
   switch (message)
   {
       case WM_INITDIALOG:
               return (TRUE);

       case WM_COMMAND:
               if (   LOWORD(wParam) == IDOK
                   || LOWORD(wParam) == IDCANCEL)
               {
                      EndDialog(hDlg, TRUE);
                      return (TRUE);
               }
               break;
   }

   return (FALSE);
}
```

התוכנית מציגה תפריט ראשי המאפשר לך להציג קווים (ועוד שני פיקסלים), מלבנים ואליפסות. היא גם מאפשרת לאפס את החלון, כלומר למחוק את תוכנו ולאפס את המיקום הנוכחי שלו. פלט לדוגמה מובא בתרשים 11.2.



תרשים 11.2: פלט לדוגמה מהתוכנית להדגמת עבודה בגרפיקה

11.13 הבנת מצבי המיפוי ואזורי התצוגה

פונקציות הטקסט והגרפיקה של חלונות פועלות לפי יחידות לוגיות, שחלונות מתרגמת אותן ליחידות פיסיות (למשל, פיקסלים) כאשר היא מציגה עצם. האופן שבו מתבצע התרגום מיחידות לוגיות ליחידות פיסיות נקבע באמצעות **מצב המיפוי** (Mapping Mode) הנוכחי. לפי ברירת המחדל, יחידות לוגיות זהות לפיקסלים. אולם באפשרותך לשנות את היחס בין יחידות לוגיות ליחידות פיסיות על ידי שינוי מצב המיפוי.

נוסף לשינוי האופן שבו חלונות ממפה את הפלט אל חלון נתון, תוכל להגדיר שתי תכונות נוספות המשפיעות על התרגום של יחידות לוגיות ליחידות פיסיות. ראשית, תוכל להגדיר את אורך ורוחב החלון במונחים של יחידות לוגיות שאתה תקבע. שנית, תוכל לקבוע את המימדים הפיסיים של אזור התצוגה. **אזור התצוגה** (Viewport) הוא אזור מסוים המתקיים בתוך גבולות החלון. מרגע שהוגדר אזור התצוגה, כל הפלט יוגבל לגבולות אלה.

בסעיף זה נבחן את הפונקציות המאפשרות לקבוע את מצב המיפוי, את מימדי החלון ואת גבולות התצוגה.

קביעת מצב המיפוי

כדי לקבוע את מצב המיפוי הנוכחי, השתמש בפונקציה ()SetMapMode, שהגדרתה היא:

```
int SetMapMode(HDC hdc, int mode);
```

הפרמטר hdc מכיל את הידית להקשר ההתקן. הפרמטר mode מגדיר את מצב המיפוי, ויכול להיות אחד הקבועים שלהלן:

מצב המיפוי	הפעולה
MM_ANISOTROPIC	הופך יחידות לוגיות ליחידות שהגדרתן נתונה בידי המתכנת, עם צירים בעלי קנה מידה שרירותי.
MM_HIENGLISH	הופך כל יחידה לוגית ליחידה פיסית בת 0.001 אינץ'.
MM_HIMETRIC	הופך כל יחידה לוגית ליחידה פיסית בת 0.01 מ"מ.
MM_ISOTROPIC	הופך יחידות לוגיות ליחידות שהגדרתן נתונה בידי המתכנת, עם צירים בקנה מידה מושווה (הדבר יוצר יחס רוחב/גובה של אחד-לאחד).
MM_LOMETRIC	הופך כל יחידה לוגית ליחידה פיסית בת 0.1 מ"מ.
MM_LOENGLISH	הופך כל יחידה לוגית ליחידה פיסית בת 0.01 אינץ'.
MM_TEXT	הופך כל יחידה לוגית לפיקסל אחד של ההתקן.
MM_TWIPS	הופך כל יחידה לוגית לאחד חלקי עשרים של נקודת דפסים, או 1/1440 האינץ', בקירוב.

הפונקציה ()SetMapMode מחזירה את מצב המיפוי הקודם, או ערך אפס במקרה של שגיאה. לפי ברירת המחדל, מצב המיפוי הוא MM_TEXT.

אפשר לשנות את מצב המיפוי מכמה סיבות. ראשית, אם אתה רוצה שהקלט של התוכנית יוצג ביחידות פיסיות, באפשרותך לבחור את אחד המצבים המשקפים את העולם הממשי, כגון MM_LOMETRIC. שנית, ייתכן שתרצה להגדיר לתוכנית שלך את היחידות המתאימות ביותר לאופי התמונה שאתה מציג. שלישית, אולי תרצה לשנות את קנה המידה של התמונה המוצגת (כלומר, ייתכן שתרצה להגדיל, או להקטין את תצוגת הפלט). לבסוף, ייתכן שתרצה להגדיר יחס גובה/רוחב של אחד-לאחד בין ציר X לציר Y. לאחר שתעשה זאת, כל יחידת X תייצג אותו מרחק פיסי כמו יחידת Y.

✍ **הערה:** **זכור**: שינוי מצב המיפוי משנה את אופן התרגום של יחידות לוגיות ליחידות פיסיות (פיקסלים).

כיצד להגדיר את גבולות החלון

בחירה במצב המיפוי MM_ISOTROPIC או MM_ANISOTROPIC תאפשר לך להגדיר את גודל החלון במונחים של יחידות לוגיות. למעשה, כאשר אתה בוחר אחד ממצבי המיפוי הללו, אתה חייב להגדיר את מימדי החלון (מצבי המיפוי הללו פועלים לפי יחידות שהוגדרו בידי המתכנת, ולכן מבחינה טכנית גבולות החלון אינם מוגדרים עד אשר תגדיר אותם). כדי להגדיר את מימדי החלון באמצעות אורך ציר X וציר Y, השתמש בפונקציה ()SetWindowExtEx, שהגדרתה היא:

```
BOOL SetWindowExtent(HDC hdc, int Xextent,
                              int Yextent, LPSIZE size);
```

הפרמטר hdc מציג את הידית להקשר ההתקן. Xextent ו-Yextent מגדירים את האורך האופקי והאנכי החדשים, הנמדדים ביחידות לוגיות. המימדים הקודמים של החלון נשמרים במבנה SIZE, ש-size הוא המצביע שלו. אם הפרמטר size מכיל ערך NULL, אזי התוכנית מתעלמת מהמימדים הקודמים. הפונקציה מחזירה ערך לא-אפס אם סיימה בהצלחה, וערך אפס אם לא. הפונקציה SetWindowExtEx() משפיעה רק כאשר מצב המיפוי הוא MM_ANISOTROPIC, או MM_ISOTROPIC. הגדרת מבנה SIZE היא:

```
typedef struct tagSIZE {
  LONG cx;
  LONG cy;
} SIZE;
```

זכור, שינוי המימדים הלוגיים של חלון אינו גורם לשינוי גודלו הפיסי על המסך. בפעולה זו אתה פשוט מגדיר את גודל החלון במונחים של היחידות הלוגיות שבחרת (למעשה, אתה מגדיר את היחס בין היחידות הלוגיות המשמשות את החלון, והיחידות הפיסיות [פיקסלים] המשמשות את ההתקן). לדוגמה, אפשר להגדיר את מימדיו של חלון כ- 100x100, או 75x50. ההבדל הוא היחס בין יחידות לוגיות לפיסקלים בעת שתמונה מוצגת על המסך.

כיצד להגדיר אזור תצוגה

כפי שהכבר הזכרנו, **אזור תצוגה** (Viewport) הוא אזור מוגדר בתוך חלון, המקבל פלט. תוכל להגדיר את מימדי אזור התצוגה באמצעות הפונקציה SetViewportExtEx(), שהגדרתה הכללית היא:

```
BOOL SetViewportExtEx(HDC hdc, int Xextent,
                               int Yextent, LPSIZE size);
```

הפרמטר hdc מכיל את הידית להקשר ההתקן. הפרמטרים Xextent ו-Yextent מגדירים ביחידות פיקסל את מימדי הציר האופקי והאנכי של אזור התצוגה. הפונקציה מחזירה ערך לא-אפס אם סיימה בהצלחה, וערך אפס אם לא. מימדיו הקודמים של אזור התצוגה מוחזרים במבנה SIZE, שהפרמטר size מצביע עליו. אם size מכיל ערך NULL, אזי התוכנית מתעלמת מהמימדים הקודמים. לפונקציה SetViewExtEx() תהיה השפעה רק כאשר מצב המיפוי הוא MM-ANISOTROPIC, או MM_ISOTROPIC.

תוכל לקבוע אזור התצוגה כל גודל הרצוי לך. כלומר, הוא יכול להשתרע על כל שטח החלון, או לתפוס רק חלק משטחו. עבור מצב המיפוי MM_TEXT, שהוא ברירת המחדל, גודל אזור התצוגה וגודל החלון זהים.

הפלט ממופה באופן אוטומטי מהקשר ההתקן של החלון (ביחידות לוגיות) אל אזור התצוגה (פיקסלים), וקנה המידה משתנה בהתאם. לפיכך, על ידי שינוי מימדי X ו-Y של אזור התצוגה, למעשה אתה משנה גודל של כל תמונה שתוצג בתוך תחום זה. כך, שאם אתה מגדיל את מימדי אזור התצוגה, יגדל בהתאם גם תוכנו. לחילופין, אם תקטין את המימדים, יקטן בהתאם תוכנו של אזור התצוגה. דבר זה נראה בתוכנית לדוגמה הבאה.

קביעת נקודת המוצא של אזור התצוגה

לפי ברירת המחדל, נקודת המוצא של אזור התצוגה היא הנקודה 0,0 בחלון. אפשר לשנות אותה באמצעות הפונקציה ()SetViewportOrgEx, שהגדרתה היא:

```
BOOL SetViewportOrgEx(HDC hdc, int X, int Y, LPPOINT OldOrg);
```

הידית של הקשר ההתקן מועברת בפרמטר hdc. נקודת המוצא החדשה לאזור התצוגה, כשהוא נתון בפיקסלים, מועבר בפרמטרים X,Y. המוצא הקודם מוחזר במבנה POINT, שהפרמטר OldOrg מצביע עליו. אם פרמטר זה מכיל ערך NULL, התוכנית מתעלמת מנקודת המוצא הקודמת.

החלפת נקודת המוצא של אזור התצוגה לובשת אופי שונה כאשר מדובר בתמונות המוצגות בחלון. תוכל לראות את השפעת הדבר בתוכנית לדוגמה המובאת כאן.

תוכנית לדוגמה לקביעת מצב המיפוי

התוכנית **Viewport11** הבאה היא גירסה מורחבת של התוכנית הגרפית הקודמת, והיא כוללת הגדרת מצב המיפוי, מימדי החלון ומימדי אזור התצוגה. פלט לדוגמה מתוכנית זו מוצג בתרשים 11.3. התוכנית מגדירה מצב מיפוי MM_ANISOTROPIC, קובעת את מימדי החלון ל- 200x200, ואת גודלו הראשוני של אזור התצוגה ל- 40X40. כשתריץ את התוכנית, כל פעם שתבחר באפשרות Magnify מתוך התפריט הראשי, יוגדל כל אחד מצירי אזור התצוגה בעשר יחידות, דבר שיגרום להגדרת התמונה בתוך החלון. בחירת האפשרות Origin מתוך התפריט תגרום להזזת נקודת המוצא של אזור התצוגה ב-50 פיקסלים, בכיוון ציר X, ובכיוון ציר Y.

```
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include "Viewport11.h"
#include "resource.h"

/* Set the mapping mode, the window and the viewport extents. */

#if defined (WIN32)
                        #define IS_WIN32 TRUE
#else
                        #define IS_WIN32 FALSE
#endif

#define IS_NT       IS_WIN32 && (BOOL)(GetVersion() < 0x80000000)
#define IS_WIN32S   IS_WIN32 && (BOOL)(!(IS_NT) &&
                    (LOBYTE(LOWORD(GetVersion()))<4))
#define IS_WIN95    (BOOL)(!(IS_NT) && !(IS_WIN32S)) && IS_WIN32
```

```
HINSTANCE hInst;   // current instance
BOOL RegisterWin95( CONST WNDCLASS* lpwc );

LPCTSTR lpszAppName  = "MyApp";
LPCTSTR lpszTitle    = "Viewport11 פרק 11 ";
char str[255]; /* holds output strings */

int X=0, Y=0; /* current output location */
int maxX, maxY; /* screen dimensions */

int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                 hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
{
   MSG      msg;
   HWND     hWnd;
   WNDCLASS wc;

   // Register the main application window class.
   //.........................................
   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = (WNDPROC)WndProc; /* window function */
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInstance;  /* handle to this instance */
   wc.hIcon         = LoadIcon( hInstance, lpszAppName );
                      /* icon style */
   wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
                      /* cursor style */
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName; /* window class name */

   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );
```

```
    hInst = hInstance;

   // Create the main application window.
   //...................................
   hWnd = CreateWindow( lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                      );

   if ( !hWnd )
      return( FALSE );

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );

   while( GetMessage( &msg, NULL, 0, 0) )
   {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }

   return( msg.wParam );
}

BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style         = lpwc->style;
   wcex.lpfnWndProc   = lpwc->lpfnWndProc;
   wcex.cbClsExtra    = lpwc->cbClsExtra;
   wcex.cbWndExtra    = lpwc->cbWndExtra;
   wcex.hInstance     = lpwc->hInstance;
   wcex.hIcon         = lpwc->hIcon;
```

```
   wcex.hCursor       = lpwc->hCursor;
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;

   // Added elements for Windows 95.
   //..............................
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");

   return RegisterClassEx( &wcex );
}

/* This function is called by Windows 95 and is passed
   messages from the message queue.*/
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam )
{
   HDC hDC;
   PAINTSTRUCT paintstruct;
   static HPEN hRedpen, hGreenpen, hBluepen, hYellowpen, hOldpen;

   static HDC memdc; /* store the virtual device handle */
   static HBITMAP hBit; /* store the virtual bitmap */
   static HBRUSH hBrush, hOldbrush; /* store the brush handle */
   static int orgX = 0, orgY = 0;

   switch( uMsg )
   {
         case WM_CREATE:
         maxX = GetSystemMetrics(SM_CXSCREEN);
         maxY = GetSystemMetrics(SM_CYSCREEN);

         /* make a compatible memory image device */
         hDC = GetDC(hWnd);
         memdc = CreateCompatibleDC(hDC);
         hBit = CreateCompatibleBitmap(hDC, maxX, maxY);
         SelectObject(memdc, hBit);
         hBrush = (HBRUSH)GetStockObject(WHITE_BRUSH);
         SelectObject(memdc, hBrush);
         PatBlt(memdc, 0, 0, maxX, maxY, PATCOPY);
```

```
        hRedpen = CreatePen(PS_SOLID, 1, RGB(255,0,0));
        hGreenpen = CreatePen(PS_SOLID, 2, RGB(0,255,0));
        hBluepen = CreatePen(PS_SOLID, 3, RGB(0,0,255));
        hYellowpen = CreatePen(PS_SOLID, 4, RGB(255, 255, 0));

        /* save default pen */
        hOldpen = (HPEN)SelectObject(memdc, hRedpen);
        SelectObject(memdc, hOldpen);

        ReleaseDC(hWnd, hDC);
                X = 40;
                Y = 40;
        break;
    case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                    case IDM_LINES:
                    /* set 2 pixels */
                    SetPixel(memdc, 40, 14, RGB(0, 0, 0));
                    SetPixel(memdc, 40, 15, RGB(0, 0, 0));

                    LineTo(memdc, 100, 50);
                    MoveToEx(memdc, 100, 50, NULL);

                    /* change to green pen */
                    hOldpen = (HPEN)SelectObject(memdc,
                               hGreenpen);
                    LineTo(memdc, 200, 100);

                    /* change to yellow pen */
                    SelectObject(memdc, hYellowpen);
                    LineTo(memdc, 0, 200);

                    /* change to blue pen */
                    SelectObject(memdc, hBluepen);
                    LineTo(memdc, 200, 200);

                    /* change to red pen */
                    SelectObject(memdc, hRedpen);
                    LineTo(memdc, 0, 0);
```

```
        /* return to default pen */
        SelectObject(memdc, hOldpen);

        Arc(memdc, 0, 0, 300, 300, 0, 50, 200, 50);

       /* show intersecting lines that define arc */
        MoveToEx(memdc, 150, 150, NULL);
        LineTo(memdc, 0, 50);
        MoveToEx(memdc, 150, 150, NULL);
        LineTo(memdc, 200, 50);

        InvalidateRect(hWnd, NULL, 1);
        break;
     case IDM_RECTANGLES:
        /* display, but don't fill */
        hOldbrush = (HBRUSH)SelectObject(memdc,
                    GetStockObject(HOLLOW_BRUSH));

        /* draw some rectangles */
        Rectangle(memdc, 50, 50, 300, 300);
        RoundRect(memdc, 125, 125, 220, 240, 15,
                 13);

        /* use a red pen */
        SelectObject(memdc, hRedpen);
        Rectangle(memdc, 100, 100, 200, 200);
        SelectObject(memdc, hOldpen);
        /* return to default pen */

        /* restore default brush */
        SelectObject(memdc, hOldbrush);

        InvalidateRect(hWnd, NULL, 1);
        break;
     case IDM_ELLIPSES:
        /* make blue brush */
        hBrush = CreateSolidBrush(RGB(0, 0, 255));
        hOldbrush = (HBRUSH)SelectObject(memdc,
                    hBrush);
```

```
/* fill these ellipses with blue */
Ellipse(memdc, 50, 200, 100, 280);
Ellipse(memdc, 75, 25, 280, 100);

/* use a red pen and fill with green */
SelectObject(memdc, hRedpen);
DeleteObject(hBrush); /* delete brush */

/* create green brush */
hBrush = CreateSolidBrush(RGB(0, 255, 0));
SelectObject(memdc, hBrush);
/* select green brush */
Ellipse(memdc, 100, 100, 200, 200);

/* draw a pie slice */
Pie(memdc, 200, 200, 340, 340, 225, 200,
    200, 250);

SelectObject(memdc, hOldpen);
/* return to default pen */
SelectObject(memdc, hOldbrush);
/* select default brush */
DeleteObject(hBrush);
/* delete green brush */
InvalidateRect(hWnd, NULL, 1);
break;
          case IDM_SIZE:
             X += 10;
             Y += 10;
             InvalidateRect(hWnd, NULL, 1);
             break;
          case IDM_ORG:
             orgX += 50;
             orgY += 50;
             InvalidateRect(hWnd, NULL, 1);
             break;
          case IDM_RESET:
             orgX = 0;
             orgY = 0;
             X = 40;
             Y = 40;
```

```
                        case IDM_CLEAR:
                /* reset current position to 0,0 */
                MoveToEx(memdc, 0, 0, NULL);

                /* erase by repainting background */
                PatBlt(memdc, 0, 0, maxX, maxY, PATCOPY);
                InvalidateRect(hWnd, NULL, 1);
                break;
              case IDM_ABOUT :
                   DialogBox( hInst, "AboutBox", hWnd,
                              (DLGPROC)About );
                   break;

              case IDM_EXIT :
                   DestroyWindow( hWnd );
                   break;
            }
            break;
    case WM_PAINT: /* process a repaint request */
         hDC = BeginPaint(hWnd, &paintstruct); /* get DC */

         /* set mapping mode, window and viewport extents */
                   SetMapMode(hDC, MM_ANISOTROPIC);
                   SetWindowExtEx(hDC, 200, 200, NULL);
                   SetViewportExtEx(hDC, X, Y, NULL);
                   SetViewportOrgEx(hDC, orgX, orgY, NULL);

         /* now, copy memory image onto screen */
         BitBlt(hDC, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
         EndPaint(hWnd, &paintstruct); /* release DC */
         break;
    case WM_DESTROY :
                   DeleteDC(memdc);
                   /* delete the memory device */
         PostQuitMessage(0);
         break;
    default :
         return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
  }

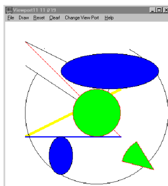
  return( 0L );
}
```

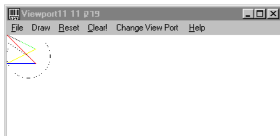
```
LRESULT CALLBACK About( HWND hDlg,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
   switch (message)
   {
       case WM_INITDIALOG:
               return (TRUE);

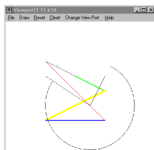
       case WM_COMMAND:
               if (   LOWORD(wParam) == IDOK
                   || LOWORD(wParam) == IDCANCEL)
               {
                        EndDialog(hDlg, TRUE);
                        return (TRUE);
               }
               break;
   }

   return (FALSE);
}
```

התוכנית השלמה נמצאת בתקליטור Chap11\Viewport11







תרשים 11.3: פלט לדוגמה מתוכנית לקביעת מצב המיפוי, הגדלות ונקודות מוצא שונות

בפרק הבא נשוב לנושא הפקדים, ונתחיל בבדיקת הפקדים הנפוצים של חלונות 9x.

פרק 12

פקדים משותפים

בפרק זה יוצג אחד הרכיבים המלהיבים ביותר של חלונות 9x: **פקדים משותפים** (Common Controls). בפרקים הקודמים למדת אודות הפקדים התקניים שנתמכים על ידי חלונות 9x וגרסאות מוקדמות יותר של חלונות. הפקדים החדשים משפרים מאוד את ממשקי היישומים השונים מבחינה תפקודית וחזותית כאחד. הם משלימים את הפקדים התקניים ומקנים למחשב גמישות ויתר עוצמה. בעזרתם נראה היישום כאילו "נתפר" במיוחד לצרכיך.

הטבלה הבאה מתארת את הפקדים המשותפים בהם תומכת חלונות 9x:

טבלה 12.1: תיאור הפקדים המשותפים בהם תומכת חלונות 9x:

פקד	תיאור
תיבות גרירת רשימות	תיבת רשימה שמאפשרת גרירה של פריטים.
פקדי כותרת	כותרות טורים.
פקדי "מקשים חמים"	תומכים במקשים חמים (קיצורי דרך) שיוצר המשתמש.
רשימות של תמונות	רשימה של תמונות גרפיות.
פקדי תצוגת רשימות	רשימת סמלים וטבלאות.
סרגלי מצב	אמצעים חזותיים לציון מצב השלמת המשימה.
גליונות תכונות	תיבת דו-שיח של תכונות.
פקדי עריכה מתקדמים	תיבת עריכה מתוחכמת.
חלונות סטטוס	סרגל המציג מידע שקשור ליישום.
פקדי כרטיסיה	תפריט מבוסס-כרטיסיה (דומה ללשוניות של תיקיות קבצים).
סרגלי כלים	תפריט מבוסס על גרפיקה.

פקד	תיאור
תווית הלחצן	תיבות טקסט זעירות שנפרשות במקום בו מוצב הסמן. מיועדות בדרך כלל לתיאור לחצנים בסרגל הכלים.
סרגלי עקיבה	פקדים שמבוססים על זחלן (מזכירים פס גלילה, אך דומים לפקד העוצמה של מערכת סטריאו).
פקדי תצוגת עץ	תצוגה במבנה של עץ.
פקדי למעלה/למטה (Spin)	חיצים בכיוון מעלה ומטה. כשהם קשורים לתיבת עריכה הם נקראים פקדי spin.

פקדים אלה נקראים **פקדים משותפים** (Common Controls), מכיון שהם מייצגים קבוצה גדולה שמנוצלת על ידי מספר גדול של יישומים. בוודאי פגשת בפקדים אלה או בחלקם במהלך עבודתך בחלונות.

פרק זה עוסק בתיאוריה וביישום של פקדים משותפים בתוכניות. בנוסף הוא מטפל בסרגלי כלים ובתוויות הלחצן (בהמשך נעסוק בפקדים משותפים נוספים).

12.1 הכללה ואתחול של פקדים משותפים

בטרם תוכל להשתמש בפקדים המשותפים, עליך לכלול בתוכנית את קובץ הכותר התקני commctrl.h. בנוסף, עליך לוודא שספריית הפקדים המשותפים מקושרת אל התוכנית שלך (בעת כתיבת הספר, ספרייה זו נקראה COMCTL32.LIB, אך מומלץ שתעיין בתיעוד של המהדר שברשותך).

יישומים העושים שימוש בפקד משותף אחד או יותר, חייבים לקרוא תחילה לפונקציה InitCommonControl(). פונקציה זו מבטיחה את טעינת ספריית DLL של הפקדים המשותפים מהדיסק, ואת אתחול תת-מערכת הפקדים המשותפים.

להלן משפט ההגדרה לפונקציה:

```
void InitCommonControl(void);
```

המקום המתאים להצבת הקריאה לפונקציה זו הוא אחרי המחלקה של החלון הראשי בתוכנית.

פקדים משותפים הם חלונות

בטרם תמשיך, חשוב להבין שכל הפקדים המשותפים הם **חלונות-בנים**. יוצרים אותם באחת משלוש הדרכים הבאות: על ידי קריאה לפונקציה CreateWindow(), על ידי קריאה לפונקציה CreateWindowEx(), או על ידי קריאה לפונקציית ממשק תכנות יישומים (API) ייחודית לפקד (הפונקציה CreateWindowEx() מאפשרת לציין מאפייני סגנון רבים). פקדים משותפים הם למעשה חלונות, לכן ניתן לנהל אותם באופן דומה לזה של חלונות אחרים בתוכנית.

פקדים משותפים רבים שולחים לתוכנית הודעות מסוג WM_COMMAND, או WM_NOTIFY, כשהמשתמש מפעיל אותם. במקרים רבים מגיבה התוכנית על ידי הודעת פקודה באמצעות פונקציית API שנקראת ()SendMessage. להלן ההגדרה לפונקציה זו:

```
LRESULT SendMessage(HWND hwnd, UINT Msg,
                    WPARAM wParam, LPARAM lParam);
```

במקרה זה, hwnd היא ידית הפקד, Msg היא ההודעה שברצונך לשלוח אל הפקד, והפרמטרים wParam ו-lParam מכילים מידע נוסף שקשור להודעה. הפונקציה מחזירה תגובה מהפקד, אם יש כזו.

סרגל הכלים

הפקד המשותף הנפוץ ביותר הוא **סרגל הכלים** (Toolbar). רכיב זה הוא במהותו תפריט גרפי שאפשרויות הבחירה שלו מיוצגות על ידי סמלים בצורת לחצנים. לעיתים קרובות מתלווה לסרגל הכלים תפריט תקני המהווה אמצעי חלופי להפעלת אפשרויות שונות.

ליצירת סרגל כלים, הפעל את הפונקציה ()CreateToolbarEx, לפי דוגמה זו:

```
HWND CreateToolbarEx(HWND hwnd, DWORD dwStyle, WORD ID,
                     int NumButtons, HINSTANCE hInst,
                     WORD BPID, LPCTBBUTTON Buttons,
                     int NumButtons,
                     int ButtonWidth, int ButtonHeight,
                     int BMPWidth, int BMPHeight,
                     UINT Size);
```

במקרה זה, hwnd היא הידית של חלון האב שסרגל הכלים שייך לו.

הסגנון של סרגל הכלים מועבר באמצעות dwStyle. סגנון סרגל הכלים חייב לכלול את WS_CHILD, ויש אפשרות שיכלול גם סגנונות כגון WS_BORDER או WS_VISIBLE. קיימים שני סגנונות ייחודיים לסרגל כלים שתוכל לכלול. הסגנון TBSTYLE_TOLTIPS למשל, מאפשר להציג תוויות לחצן לרכיבי ההפעלה השונים (בהמשך הפרק נרחיב בנושא התוויות). הסגנון TBSTYLE_WRAPABLE מאפשר להציג סרגל כלים ארוך.

הפרמטר ID מעביר את **המזהה** (Identifier) שקשור לסרגל הכלים. הפרמטר NumButtons מעביר את מספר הלחצנים שבסרגל הכלים. הפרמטר hInst מעביר את ידית המופע של היישום. הפרמטר BPID מעביר את המזהה של משאב מפת-הסיביות שיוצרת את סרגל הכלים.

Buttons מצביע אל ה**מבנים** (Structures) של המערך TBBUTTON, שאליו מועברים נתוני הלחצנים. NumButtons מציין את מספר הלחצנים שבסרגל הכלים. ButtonWidth מכיל את הרוחב של הלחצן, ו-ButtonHeight מכיל את גובהו. BMPWidth מכיל את רוחב הסמל של הלחצן, ו-BMPHeight מכיל את גובהו. אם משתני הרוחב והגובה של הלחצן מכילים את הערך 0, התוכנית תתאים בעצמה את מימדי הלחצן לגודל מפת-הסיביות הנתונה. Size מכיל את גודל המבנה של TBBUTON.

הפונקציה מחזירה ידית לחלון של סרגל הכלים.

לכל לחצן קשור מבנה TBBUTTON אשר מגדיר את תכונותיו השונות. להלן המבנה TBBUTTON :

```
typedef struct _TBBUTTON {
  int iBitmap;
  int idCommand;
  BYTE fsState;
  BYTE fsStyle;
  DWORD dwData;
  int iString;
} TBBUTTON;
```

iBitmap מכיל את האינדקס של תמונת מפת-הסיביות של הלחצן. האינדקס של הלחצנים מתחיל ב-0, והם מוצגים משמאל לימין.

הפקודה הקשורה ללחצן נמצאת ב-idCommand. הלחיצה על הלחצן יוצרת הודעת WM_COMMAND אשר נשלחת אל חלון האב. הערך המספרי של idCommand נשמר בחלק הנמוך של המילה wParam.

המשתנה fsState מכיל את המצב ההתחלתי של הלחצן. הוא יכול להיות אחד, או יותר מהערכים שמציגה הטבלה הבאה :

טבלה 12.2: הערכים של המשתנה fsState

מצב	משמעות
TBSTATE_CHECKED	לחצן לחוץ
TBSTATE_ENABLE	לחצן פעיל
TBSTATE_HIDDEN	לחצן מוסתר ובלתי פעיל
TBSTATE_INDETERMINATE	לחצן אפור ובלתי פעיל
TBSTATPRESSEDE_	לחצן לחוץ
TBSTATE_WRAP	הלחצנים הבאים נמצאים בשורה חדשה

המשתנה fsStyle מכיל את סגנון הלחצן, ויכול לקבל צירוף חוקי כלשהו מתוך רשימת הערכים הבאים:

טבלה 12.3: הערכים של המשתנה fsStyle

סגנון	משמעות
TBSTYLE_BUTTON	לחצן תקני
TBSTYLE_CHECK	הלחצן מחליף את מצבו ממסומן ללא-מסומן ולהיפך, בכל פעם שלוחצים עליו
TBSTYLE_CHECKGROUP	לחצן סימון שהוא חלק **מקבוצה בלעדית** (Mutually Exclusive)
TBSTYLE_GROUP	לחצן תקני שהוא חלק מקבוצה בלעדית
TBSTYLE_SEP	מפריד לחצנים (בסגנון idCommand חייב לקבל את הערך 0)

שים לב לסגנון TBSTYLE_SEP. סגנון זה משמש לקביעת מרווח בין לחצנים בסרגל הכלים, ומאפשר להפריד את הלחצנים לקבוצות.

השדה dwData מכיל נתונים המוגדרים על ידי המשתמש. השדה iString הוא אינדקס המחרוזת הקשורה ללחצן. אם אין ברצונך להשתמש בו, קבע לו את הערך 0.

תצורת המחדל של סרגלי הכלים מאפשרת להם לפעול באופן אוטומטי לחלוטין ללא כל צורך בהתערבות מצד התוכנית. יחד עם זאת, ניתן לטפל בהם ידנית באמצעות שיגור **הודעות בקרה** (Control Messages) מפורשות בעזרת הפונקציה ()SendMessage. להלן שלוש הודעות שכיחות:

טבלה 12.4: הודעות בקרה (Control Messages)

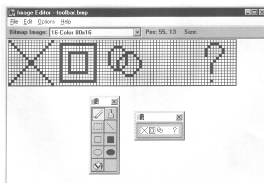
הודעה	משמעות
TB_CHECKBUTTON	לחיצה או מחיקה של לחצן. wParam חייב להכיל את מספר הזיהוי של הלחצן. lParam יקבל ערך שונה מ-0 עבור לחיצה, או 0 במקרה של מחיקת לחצן.
TB_ENABLEBUTTON	קביעת מצב הפעילות של הלחצן. wParam חייב להכיל את מספר הזיהוי של הלחצן. lParam יקבל ערך שונה מ-0 להפעלת הלחצן, או 0 לביטול הפעלתו.
TB_HIDEBUTTON	הצגה או הסתרה של לחצן. wParam חייב להכיל את מספר הזיהוי של הלחצן. lParam יקבל ערך שונה מ-0 להסתרת הלחצן, או 0 להצגתו.

סרגלי הכלים גם מסוגלים ליצור הודעות שתפקידן להסב את תשומת ליבה של התוכנית לפעולות שונות הקשורות בסרגל הכלים. אם סרגל הכלים פשוט, אין צורך להתייחס להודעות אלו (כל ההודעות מסוג זה מתחילות במחרוזת _TBN, ניתן ללמוד עליהן באמצעות קובץ הכותר commctrl.h, או מתוך תיעוד ספריית ממשק תכנות היישומים, API).

יצירת מפת-סיביות של סרגל כלים

כדי להשתמש בסרגל הכלים עליך ליצור תחילה את התמונות הגרפיות שמכילים הלחצנים, בעזרת עורך הדמות. תהליך יצירת הגרפיקה של הלחצן דומה ליצירת סמל בודד. עם זאת, יש להביא בחשבון נקודה חשובה: קיימת רק מפת-סיביות אחת שקשורה לסרגל הכלים, ועליה להכיל את **כל** דמויות הלחצנים. מכאן, אם סרגל הכלים מכיל שישה לחצנים, אזי מפת-הסיביות שלו חייבת להגדיר שש דמויות. לדוגמה, מפת-הסיביות של סרגל כלים שמכיל שישה לחצנים שכל אחד מהם בגודל 16x16 סיביות, תהיה בגובה של 16 סיביות ובאורך של 96 סיביות (6x16).

סרגלי הכלים המוצגים בפרק זה כוללים שש דמויות, כל אחת בגודל 16x15 סיביות. כלומר, יהיה עליך ליצור מפת-סיביות בגודל 16x96 סיביות. תרשים 12.1 מראה כיצד מציג עורך הדמויות את מפת-הסיביות שיוצרת תוכנית ההדגמה שמובאת בפרק זה. סרגל הכלים ישמש כתפריט חלופי עבור תוכנית הגרפיקה המופיעה בפרק 11. שמור את מפת-הסיביות בקובץ בשם TOOLBAR.BMP.



תרשים 12.1: מפת-הסיביות של סרגל הכלים במצב עריכה

תוכנית לדוגמה של סרגל כלים פשוט

התוכנית **Viewport12** הבאה (נמצאת בתקליטור Chap12\Viewport12) מוסיפה סרגל כלים לתוכנית הגרפיקה שבפרק 11. סרגל הכלים מכפיל את אפשרויות התפריט, ומאפשר להציג שורות, מרובעים ואליפסות. ניתן גם להזיז את ראשית הצירים, להגדיל את התמונה ולהציג תפריט עזרה. להלן התוכנית של סרגל הכלים:

```
#include <windows.h>
#include <commctrl.h>
#include <string.h>
#include <stdio.h>
#include "Viewport12.h"
#include "resource.h"

/* Set the mapping mode, the window and the viewport extents. */

#if defined (WIN32)
                        #define IS_WIN32 TRUE
#else
                        #define IS_WIN32 FALSE
#endif

#define IS_NT      IS_WIN32 && (BOOL)(GetVersion() < 0x80000000)
#define IS_WIN32S  IS_WIN32 && (BOOL)(!(IS_NT) &&
                   (LOBYTE(LOWORD(GetVersion()))<4))
#define IS_WIN95   (BOOL)(!(IS_NT) && !(IS_WIN32S)) && IS_WIN32

HINSTANCE hInst;  // current instance
BOOL RegisterWin95( CONST WNDCLASS* lpwc );
void InitToolbar();

LPCTSTR lpszAppName  = "MyApp";
LPCTSTR lpszTitle    = "פרק 12 Viewport12 ";
char str[255]; /* holds output strings */

int X=0, Y=0; /* current output location */
int maxX, maxY; /* screen dimensions */
HWND tbWnd;
TBBUTTON tbButtons[NUMBUTTONS];
```

```
int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                  hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
{
   MSG      msg;
   HWND     hWnd;
   WNDCLASS wc;

   // Register the main application window class.
   //.........................................
   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = (WNDPROC)WndProc; /* window function */
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInstance;  /* handle to this instance */
   wc.hIcon         = LoadIcon( hInstance, lpszAppName );
                      /* icon style */
   wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
                      /* cursor style */
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName; /* window class name */

   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );

    hInst = hInstance;

   // Create the main application window.
   //..................................
   hWnd = CreateWindow( lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
```

```
                        NULL,
                        hInstance,
                        NULL
                      );

   if ( !hWnd )
      return( FALSE );

   InitToolbar();
   InitCommonControls();
   tbWnd = CreateToolbarEx(hWnd,
                           WS_VISIBLE | WS_CHILD | WS_BORDER,
                           ID_TOOLBAR,
                           NUMBUTTONS,
                           hInstance,
                           IDB_TOOLBAR,
                           tbButtons,
                           NUMBUTTONS,
                           0,0,16,16,
                           sizeof(TBBUTTON));

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );

   while( GetMessage( &msg, NULL, 0, 0) )
   {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }

   return( msg.wParam );

}

BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style         = lpwc->style;
   wcex.lpfnWndProc   = lpwc->lpfnWndProc;
```

```
   wcex.cbClsExtra    = lpwc->cbClsExtra;
   wcex.cbWndExtra    = lpwc->cbWndExtra;
   wcex.hInstance     = lpwc->hInstance;
   wcex.hIcon         = lpwc->hIcon;
   wcex.hCursor       = lpwc->hCursor;
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;

   // Added elements for Windows 95.
   //..............................
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");

   return RegisterClassEx( &wcex );
}

/* This function is called by Windows 95 and is passed
   messages from the message queue.*/
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam )
{
   HDC hDC;
   PAINTSTRUCT paintstruct;
   static HPEN hRedpen, hGreenpen, hBluepen, hYellowpen, hOldpen;

   static HDC memdc; /* store the virtual device handle */
   static HBITMAP hBit; /* store the virtual bitmap */
   static HBRUSH hBrush, hOldbrush; /* store the brush handle */
   static int orgX = 0, orgY = 0;

   switch( uMsg )
   {
         case WM_CREATE:
         maxX = GetSystemMetrics(SM_CXSCREEN);
         maxY = GetSystemMetrics(SM_CYSCREEN);

         /* make a compatible memory image device */
         hDC = GetDC(hWnd);
         memdc = CreateCompatibleDC(hDC);
```

```
    hBit = CreateCompatibleBitmap(hDC, maxX, maxY);
    SelectObject(memdc, hBit);
    hBrush = (HBRUSH)GetStockObject(WHITE_BRUSH);
    SelectObject(memdc, hBrush);
    PatBlt(memdc, 0, 0, maxX, maxY, PATCOPY);

    hRedpen = CreatePen(PS_SOLID, 1, RGB(55,100,80));
    hGreenpen = CreatePen(PS_SOLID, 2, RGB(0,255,0));
    hBluepen = CreatePen(PS_SOLID, 3, RGB(0,0,255));
    hYellowpen = CreatePen(PS_SOLID, 4, RGB(255, 255, 0));

    /* save default pen */
    hOldpen = (HPEN)SelectObject(memdc, hRedpen);
    SelectObject(memdc, hOldpen);

    ReleaseDC(hWnd, hDC);
                X = 40;
                Y = 40;
    break;
case WM_COMMAND :
        switch( LOWORD( wParam ) )
        {
                case IDM_LINES:
                /* set 2 pixels */
                SetPixel(memdc, 40, 14, RGB(0, 0, 0));
                SetPixel(memdc, 40, 15, RGB(0, 0, 0));

                LineTo(memdc, 100, 50);
                MoveToEx(memdc, 100, 50, NULL);

                /* change to green pen */
                hOldpen = (HPEN)SelectObject(memdc,
                           hGreenpen);
                LineTo(memdc, 200, 100);

                /* change to yellow pen */
                SelectObject(memdc, hYellowpen);
                LineTo(memdc, 0, 200);
```

```
            /* change to blue pen */
            SelectObject(memdc, hBluepen);
            LineTo(memdc, 200, 200);

            /* change to red pen */
            SelectObject(memdc, hRedpen);
            LineTo(memdc, 0, 0);

            /* return to default pen */
            SelectObject(memdc, hOldpen);

            Arc(memdc, 0, 0, 300, 300, 0, 50, 200, 50);

           /* show intersecting lines that define arc */
            MoveToEx(memdc, 150, 150, NULL);
            LineTo(memdc, 0, 50);
            MoveToEx(memdc, 150, 150, NULL);
            LineTo(memdc, 200, 50);

            InvalidateRect(hWnd, NULL, 1);
            break;
         case IDM_RECTANGLES:
            /* display, but don't fill */
            hOldbrush = (HBRUSH)SelectObject(memdc,
                 (HBRUSH)GetStockObject(HOLLOW_BRUSH));

            /* draw some rectangles */
            Rectangle(memdc, 50, 50, 300, 300);
            RoundRect(memdc, 125, 125, 220, 240, 15,
                      13);

            /* use a red pen */
            SelectObject(memdc, hRedpen);
            Rectangle(memdc, 100, 100, 200, 200);
            SelectObject(memdc, hOldpen);
            /* return to default pen */

            /* restore default brush */
            SelectObject(memdc, hOldbrush);

            InvalidateRect(hWnd, NULL, 1);
            break;
```

```
case IDM_ELLIPSES:
   /* make blue brush */
   hBrush = CreateSolidBrush(RGB(0, 0, 255));
   hOldbrush = (HBRUSH)SelectObject(memdc,
               hBrush);

   /* fill these ellipses with blue */
   Ellipse(memdc, 50, 200, 100, 280);
   Ellipse(memdc, 75, 25, 280, 100);

   /* use a red pen and fill with green */
   SelectObject(memdc, hRedpen);
   DeleteObject(hBrush); /* delete brush */

   /* create green brush */
   hBrush = CreateSolidBrush(RGB(0, 255, 0));
   SelectObject(memdc, hBrush);
   /* select green brush */
   Ellipse(memdc, 100, 100, 200, 200);

   /* draw a pie slice */
   Pie(memdc, 200, 200, 340, 340, 225, 200,
       200, 250);

   SelectObject(memdc, hOldpen);
   /* return to default pen */
   SelectObject(memdc, hOldbrush);
   /* select default brush */
   DeleteObject(hBrush);
   /* delete green brush */
   InvalidateRect(hWnd, NULL, 1);
   break;
          case IDM_SIZE:
             X += 10;
             Y += 10;
              InvalidateRect(hWnd, NULL, 1);
             break;
          case IDM_ORG:
             orgX += 50;
             orgY += 50;
             InvalidateRect(hWnd, NULL, 1);
              break;
```

```
                    case IDM_RESET:
                       orgX = 0;
                       orgY = 0;
                       X = 40;
                       Y = 40;
                    case IDM_SHOW:
                       ShowWindow(tbWnd, SW_RESTORE);
                       break;
                    case IDM_HIDE:
                       ShowWindow(tbWnd, SW_HIDE);
                       break;
                    case IDM_CLEAR:
                     /* reset current position to 0,0 */
                       MoveToEx(memdc, 0, 0, NULL);

                    /* erase by repainting background */
                      PatBlt(memdc, 0, 0, maxX, maxY,
                             PATCOPY);
                       InvalidateRect(hWnd, NULL, 1);
                       break;
                    case IDM_ABOUT :
                      /* Show abour button as pressed*/
                       SendMessage(tbWnd,TB_CHECKBUTTON,
                        (LPARAM) IDM_ABOUT, (WPARAM) 1);
                       DialogBox( hInst, "AboutBox",
                                hWnd, (DLGPROC)About );

                      /* Reset the help button */
                       SendMessage(tbWnd,TB_CHECKBUTTON,
                             (LPARAM) IDM_ABOUT, MB_OK);
                        break;
                    case IDM_EXIT :
                       DestroyWindow( hWnd );
                        break;
          }
          break;
```

```
      case WM_PAINT: /* process a repaint request */
            hDC = BeginPaint(hWnd, &paintstruct); /* get DC */

            /* set mapping mode, window and viewport extents */
           SetMapMode(hDC, MM_ANISOTROPIC);
           SetWindowExtEx(hDC, 200, 200, NULL);
           SetViewportExtEx(hDC, X, Y, NULL);
            SetViewportOrgEx(hDC, orgX, orgY, NULL);

            /* now, copy memory image onto screen */
            BitBlt(hDC, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
            EndPaint(hWnd, &paintstruct); /* release DC */
            break;
      case WM_DESTROY :
           DeleteObject(hRedpen);
           DeleteObject(hGreenpen);
           DeleteObject(hBluepen);
           DeleteObject(hYellowpen);

           DeleteDC(memdc); /* delete the memory device */
           PostQuitMessage(0);
            break;
      default :
            return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
   }

   return( 0L );
}

LRESULT CALLBACK About( HWND hDlg,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
   switch (message)
   {
       case WM_INITDIALOG:
               return (TRUE);
```

```
       case WM_COMMAND:
               if (   LOWORD(wParam) == IDOK
                   || LOWORD(wParam) == IDCANCEL)
               {
                       EndDialog(hDlg, TRUE);
                       return (TRUE);
               }
               break;
   }

   return (FALSE);
}

/* Initialize the toolbar structures. */
void InitToolbar()
{
  tbButtons[0].iBitmap = 0;
  tbButtons[0].idCommand = IDM_LINES;
  tbButtons[0].fsState = TBSTATE_ENABLED;
  tbButtons[0].fsStyle = TBSTYLE_BUTTON;
  tbButtons[0].dwData = 0L;
  tbButtons[0].iBitmap = 0;
  tbButtons[0].iString = 0;

  tbButtons[1].iBitmap = 1;
  tbButtons[1].idCommand = IDM_RECTANGLES;
  tbButtons[1].fsState = TBSTATE_ENABLED;
  tbButtons[1].fsStyle = TBSTYLE_BUTTON;
  tbButtons[1].dwData = 0L;
  tbButtons[1].iString = 0;

  tbButtons[2].iBitmap = 2;
  tbButtons[2].idCommand = IDM_ELLIPSES;
  tbButtons[2].fsState = TBSTATE_ENABLED;
  tbButtons[2].fsStyle = TBSTYLE_BUTTON;
  tbButtons[2].dwData = 0L;
  tbButtons[2].iString = 0;
```

```
  /* button separator */
  tbButtons[3].iBitmap = 0;
  tbButtons[3].idCommand = 0;
  tbButtons[3].fsState = TBSTATE_ENABLED;
  tbButtons[3].fsStyle = TBSTYLE_SEP;
  tbButtons[3].dwData = 0L;
  tbButtons[3].iString = 0;

  tbButtons[4].iBitmap = 3;
  tbButtons[4].idCommand = IDM_SIZE;
  tbButtons[4].fsState = TBSTATE_ENABLED;
  tbButtons[4].fsStyle = TBSTYLE_BUTTON;
  tbButtons[4].dwData = 0L;
  tbButtons[4].iString = 0;

  tbButtons[5].iBitmap = 4;
  tbButtons[5].idCommand = IDM_ORG;
  tbButtons[5].fsState = TBSTATE_ENABLED;
  tbButtons[5].fsStyle = TBSTYLE_BUTTON;
  tbButtons[5].dwData = 0L;
  tbButtons[5].iString = 0;

  tbButtons[6].iBitmap = 0;
  tbButtons[6].idCommand = 0;
  tbButtons[6].fsState = TBSTATE_ENABLED;
  tbButtons[6].fsStyle = TBSTYLE_SEP;
  tbButtons[6].dwData = 0L;
  tbButtons[6].iString = 0;

  tbButtons[7].iBitmap = 5;
  tbButtons[7].idCommand = IDM_ABOUT;
  tbButtons[7].fsState = TBSTATE_ENABLED;
  tbButtons[7].fsStyle = TBSTYLE_BUTTON;
  tbButtons[7].dwData = 0L;
  tbButtons[7].iString = 0;
}
```

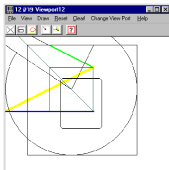
רוב הקוד בתוכנית אינו דורש הסבר נוסף. נתאר אותו בקצרה (זכור, כל שורות הקוד שאינן מתייחסות לסרגל הכלים, הוסברו בפרק 11). נתוני סרגל הכלים שמורים במערך tbButtons. הפונקציה InitToolBar() מאתחלת את המערך. שים לב שהמבנים החמישי והשביעי במערך הם למעשה, **מפריד לחצנים**. הפונקציה WinMain() קוראת לפונקציה InitCommonControls(). לאחר מכן, נוצר סרגל הכלים וידית משוייכת ל-tbWnd.

כל אחד מלחצני סרגל הכלים מקביל לאפשרות בחירה בתפריט הראשי. ליתר דיוק, כל לחצן (למעט המפריד) קשור אל מספר מזהה (ID) בתפריט. כשלוחצים על לחצן כלשהו, המספר המזהה הקשור אליו נשלח אל התוכנית כחלק מהודעת WM_COMMAND, כאילו בחרו באפשרות התפריט המקבילה. למעשה, אותו משפט case מטפל בלחיצות הלחצנים ובבחירה מתוך התפריט.

סרגל הכלים הוא חלון, ולכן ניתן להציגו, או להסתירו ככל חלון אחר באמצעות הפונקציה ShowWindow(). להסתרת החלון, בחר באפשרות Hideבתוך Toolbar (הסתר סרגל כלים) בתפריט View (אפשרויות). להצגתו מחדש, בחר Show בתוך Toolbar (הצג סרגל כלים). מאחר שסרגל הכלים חופף חלקית את אזור הלקוח בחלון הראשי, עליך לאפשר למשתמש להסיר את סרגל הכלים אם אין צורך בו. ניתן לראות בתוכנית שפשוט מאוד לעשות זאת.

ראוי לציין נקודה נוספת בתוכנית. שים לב לקוד המופיע תחת המשפט case IDM_ABOUT. כשבוחרים About (עזרה, באמצעות התפריט הראשי, או בלחיצה על לחצן Help), נשלחת ההודעה TB_CHECKBUTTON וגורמת ללחיצת הלחצן Help שבסרגל הכלים. לאחר שהמשתמש סוגר את תיבת ההודעה Help, חוזר הלחצן באופן ידני. מנגנון זה מאפשר ללחצן להישאר במצב לחוץ, כל עוד מוצגת תיבת ההודעה Help. כך הדגמנו כיצד ניתן לטפל בסרגל הכלים באופן ידני במסגרת התוכנית.

תרשים 12.2 מציג פלט לדוגמה של תוכנית סרגל הכלים.



תרשים 12.2: פלט לדוגמה של תוכנית סרגל הכלים

הוספת תוויות לחצן

בוודאי הבחנת שחלונות 9x מציגה על המסך חלונות טקסט זעירים כשסמן העכבר שוהה על גבי סרגל הכלים במשך שנייה בקירוב. חלונות אלה נקראים **תוויות לחצן** (Tooltips). תוויות הלחצן אינן חיוניות לפעולתו, אך יש לכלול אותן ברוב סרגלי הכלים, מכיון שהמשתמש מצפה לראות אותן. בקטע זה נוסיף תוויות לחצן לסרגל הכלים.

כדי להוסיף תוויות לחצן לסרגל הכלים, עליך לכלול תחילה את הסגנון TBSTYLE_TOOLTIPS בעת יצירת הסרגל. סגנון זה מאפשר שיגור הודעות WM_NOTIFY כשהסמן שוהה מעל לחצן כלשהו במשך שנייה בקירוב. עם קבלת הודעת WM_NOTIFY, יצביע lParam אל מבנה TOOLTIPTEXT, המוגדר באופן הבא:

```
typedef struct
{
  NMHDR hdr;
  LPSTR lpszText;
  char szText[80];
  HINSTANCE hinst;
  UINT uFlags;
} TOOLTIPTEXT;
```

האיבר (Member) הראשון של TOOLTIPTEXT הוא המבנה NMHDR, המוגדר כך:

```
typedef struct tagNMHDR
{
  HWND  hwndFrom; /* handle of control */
  UINT  idFrom; /* control ID */
  UINT  code; /* notification code */
} NMHDR;
```

כשמגיעה דרישה להצגת תווית לחצן, code יכיל את TTN_NEEDTEXT ו-idForm יכיל את המספר המזהה (ID) של הלחצן הנדון. קיימות שלוש דרכים להצגת תווית הלחצן המבוקשת; ניתן להעתיק את הטקסט של התווית אל המערך szText של TOOLTIPTEXT, להצביע אל הטקסט בעזרת lpszText, או לספק את המספר המזהה של משאב המחרוזת. כשבוחרים באפשרות משאב המחרוזת, lpszText מקבל את המספר המזהה של המחרוזת ו-hinst חייבת להיות הידית של משאב המחרוזת. הדרך הקלה ביותר היא להצביע בעזרת lpszText אל מחרוזת שמספקת התוכנית. לדוגמה, משפט case הבא מגיב לבקשה של תווית לחצן בתוכנית הגרפיקה.

```
    case WM_NOTIFY: /* respond to tooltip request */
          TTtext = (LPTOOLTIPTEXT) lParam;
            if (TTtext->hdr.code == TTN_NEEDTEXT)
           switch(TTtext->hdr.idFrom)
         {
            case IDM_LINES:
                     TTtext->lpszText = "Lines";
                     break;
            case IDM_RECTANGLES:
                     TTtext->lpszText = "Rectangles";
                     break;
            case IDM_ELLIPSES:
                  TTtext->lpszText = "Ellipses";
                     break;
             case IDM_ORG:
                     TTtext->lpszText = "Change Org";
                     break;
             case IDM_SIZE:
                     TTtext->lpszText = "Change size";
                     break;
         }
         break;
```

לאחר שנבחר הטקסט המתאים והשליטה חזרה למערכת ההפעלה, תוצג תווית הלחצן באופן אוטומטי. אין התוכנית צריכה לבצע פעולה נוספת כלשהי. כפי שנוכחת לדעת, השימוש בתוויות הלחצן הוא אוטומטי ופשוט מאוד להוסיפן ליישומים.

תוכנית סרגל הכלים בשלמותה, כולל תוויות לחצן

לפניך תוכנית סרגל הכלים **Viewport12_b** בשלמותה, כולל תוויות לחצן. גירסה זו מנצלת את קובץ המשאבים וקובץ הכותר ששימשו את הגירסה הקודמת. התוכנית נקראת Viewport12_b והיא נמצאת בתקליטור Chap12\Viewport12_b. תרשים 12.3 מציג פלט דוגמה של התוכנית.

```
#include <windows.h>
#include <commctrl.h>
#include <string.h>
#include <stdio.h>
#include "Viewport12_b.h"
#include "resource.h"
```

```
/* Set the mapping mode, the window and the viewport extents. */

#if defined (WIN32)
     #define IS_WIN32 TRUE
#else
     #define IS_WIN32 FALSE
#endif

#define IS_NT       IS_WIN32 && (BOOL)(GetVersion() < 0x80000000)
#define IS_WIN32S  IS_WIN32 && (BOOL)(!(IS_NT) &&
        (LOBYTE(LOWORD(GetVersion()))<4))
#define IS_WIN95    (BOOL)(!(IS_NT) && !(IS_WIN32S)) && IS_WIN32

HINSTANCE hInst;  // current instance
BOOL RegisterWin95( CONST WNDCLASS* lpwc );
void InitToolbar();

LPCTSTR lpszAppName  = "MyApp";
LPCTSTR lpszTitle    = "פרק 12 Viewport12 with tooltips ";

int X=0, Y=0; /* current output location */
int maxX, maxY; /* screen dimensions */
HWND tbWnd;
TBBUTTON tbButtons[NUMBUTTONS];

int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE
                  hPrevInstance, LPTSTR lpCmdLine, int
nCmdShow)
{
   MSG       msg;
   HWND      hWnd;
   WNDCLASS wc;

   // Register the main application window class.
   //.........................................
   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = (WNDPROC)WndProc; /* window function */
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInstance;  /* handle to this instance */
```

```
wc.hIcon         = LoadIcon( hInstance, lpszAppName );
                   /* icon style */
wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
                   /* cursor style */
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wc.lpszMenuName  = lpszAppName;
wc.lpszClassName = lpszAppName; /* window class name */

if ( IS_WIN95 )
{
   if ( !RegisterWin95( &wc ) )
      return( FALSE );
}
else if ( !RegisterClass( &wc ) )
   return( FALSE );

 hInst = hInstance;

// Create the main application window.
//....................................
hWnd = CreateWindow( lpszAppName,
                     lpszTitle,
                     WS_OVERLAPPEDWINDOW,
                     CW_USEDEFAULT, 0,
                     CW_USEDEFAULT, 0,
                     NULL,
                     NULL,
                     hInstance,
                     NULL
                   );

if ( !hWnd )
   return( FALSE );

InitToolbar();
InitCommonControls();
tbWnd = CreateToolbarEx(hWnd,
      WS_VISIBLE | WS_CHILD | WS_BORDER | TBSTYLE_TOOLTIPS,
      ID_TOOLBAR,
      NUMBUTTONS,
```

```
         hInstance,
         IDB_TOOLBAR,
         tbButtons,
         NUMBUTTONS,
         0,0,16,16,
         sizeof(TBBUTTON));

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );

   while( GetMessage( &msg, NULL, 0, 0) )
   {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }

   return( msg.wParam );
}

BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style         = lpwc->style;
   wcex.lpfnWndProc   = lpwc->lpfnWndProc;
   wcex.cbClsExtra    = lpwc->cbClsExtra;
   wcex.cbWndExtra    = lpwc->cbWndExtra;
   wcex.hInstance     = lpwc->hInstance;
   wcex.hIcon         = lpwc->hIcon;
   wcex.hCursor       = lpwc->hCursor;
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;

   // Added elements for Windows 95.
   //..............................
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.hIconSm = LoadIcon(wcex.hInstance, "SMALL");

   return RegisterClassEx( &wcex );
}
```

```
/* This function is called by Windows 95 and is passed
   messages from the message queue.*/
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam )
{
   HDC hDC;
   LPTOOLTIPTEXT TTtext;
   PAINTSTRUCT paintstruct;
   static HPEN hRedpen, hGreenpen, hBluepen, hYellowpen,
hOldpen;

   static HDC memdc; /* store the virtual device handle */
   static HBITMAP hBit; /* store the virtual bitmap */
   static HBRUSH hBrush, hOldbrush; /* store the brush handle */
   static int orgX = 0, orgY = 0;

   switch( uMsg )
   {
       case WM_CREATE:
         maxX = GetSystemMetrics(SM_CXSCREEN);
         maxY = GetSystemMetrics(SM_CYSCREEN);

         /* make a compatible memory image device */
         hDC = GetDC(hWnd);
         memdc = CreateCompatibleDC(hDC);
         hBit = CreateCompatibleBitmap(hDC, maxX, maxY);
         SelectObject(memdc, hBit);
         hBrush = (HBRUSH)GetStockObject(WHITE_BRUSH);
         SelectObject(memdc, hBrush);
         PatBlt(memdc, 0, 0, maxX, maxY, PATCOPY);

         hRedpen = CreatePen(PS_SOLID, 1, RGB(55,100,80));
         hGreenpen = CreatePen(PS_SOLID, 2, RGB(0,255,0));
         hBluepen = CreatePen(PS_SOLID, 3, RGB(0,0,255));
         hYellowpen = CreatePen(PS_SOLID, 4, RGB(255, 255, 0));

         /* save default pen */
         hOldpen = (HPEN)SelectObject(memdc, hRedpen);
         SelectObject(memdc, hOldpen);
```

```
        ReleaseDC(hWnd, hDC);
                        X = 40;
                        Y = 40;
        break;
      case WM_NOTIFY: /* respond to tooltip request */
         TTtext = (LPTOOLTIPTEXT) lParam;
                        if (TTtext->hdr.code == TTN_NEEDTEXT)
                        switch(TTtext->hdr.idFrom)
                        {
                            case IDM_LINES:
                               TTtext->lpszText = "Lines";
                               break;
                            case IDM_RECTANGLES:
                               TTtext->lpszText = "Rectangles";
                               break;
                            case IDM_ELLIPSES:
                               TTtext->lpszText = "Ellipses";
                               break;
                            case IDM_ORG:
                               TTtext->lpszText = "Change Org";
                               break;
                            case IDM_SIZE:
                               TTtext->lpszText = "Change size";
                               break;
                            case IDM_ABOUT:
                               TTtext->lpszText = "About ";
                               break;
                         }
                         break;

    case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                    case IDM_LINES:
                    /* set 2 pixels */
                    SetPixel(memdc, 40, 14, RGB(0, 0, 0));
                    SetPixel(memdc, 40, 15, RGB(0, 0, 0));
```

```
      LineTo(memdc, 100, 50);
      MoveToEx(memdc, 100, 50, NULL);

      /* change to green pen */
      hOldpen = (HPEN)SelectObject(memdc,
                hGreenpen);
      LineTo(memdc, 200, 100);

      /* change to yellow pen */
      SelectObject(memdc, hYellowpen);
      LineTo(memdc, 0, 200);

      /* change to blue pen */
      SelectObject(memdc, hBluepen);
      LineTo(memdc, 200, 200);

      /* change to red pen */
      SelectObject(memdc, hRedpen);
      LineTo(memdc, 0, 0);

      /* return to default pen */
      SelectObject(memdc, hOldpen);

      Arc(memdc, 0, 0, 300, 300, 0, 50, 200, 50);

      /* show intersecting lines that define arc */
      MoveToEx(memdc, 150, 150, NULL);
      LineTo(memdc, 0, 50);
      MoveToEx(memdc, 150, 150, NULL);
      LineTo(memdc, 200, 50);

      InvalidateRect(hWnd, NULL, 1);
      break;
    case IDM_RECTANGLES:
      /* display, but don't fill */
      hOldbrush = (HBRUSH)SelectObject(memdc,
                  GetStockObject(HOLLOW_BRUSH));
```

```
            /* draw some rectangles */
            Rectangle(memdc, 50, 50, 300, 300);
            RoundRect(memdc, 125, 125, 220, 240, 15,
                      13);

            /* use a red pen */
            SelectObject(memdc, hRedpen);
            Rectangle(memdc, 100, 100, 200, 200);
            SelectObject(memdc, hOldpen);
            /* return to default pen */

            /* restore default brush */
            SelectObject(memdc, hOldbrush);

            InvalidateRect(hWnd, NULL, 1);
            break;
         case IDM_ELLIPSES:
            /* make blue brush */
            hBrush = CreateSolidBrush(RGB(0, 0, 255));
            hOldbrush = (HBRUSH)SelectObject(memdc,
                        hBrush);

            /* fill these ellipses with blue */
            Ellipse(memdc, 50, 200, 100, 280);
            Ellipse(memdc, 75, 25, 280, 100);

            /* use a red pen and fill with green */
            SelectObject(memdc, hRedpen);
            DeleteObject(hBrush); /* delete brush */

            /* create green brush */
            hBrush = CreateSolidBrush(RGB(0, 255, 0));
            SelectObject(memdc, hBrush);
            /* select green brush */
            Ellipse(memdc, 100, 100, 200, 200);
```

```
/* draw a pie slice */
Pie(memdc, 200, 200, 340, 340, 225, 200,
    200, 250);

SelectObject(memdc, hOldpen);
/* return to default pen */
SelectObject(memdc, hOldbrush);
/* select default brush */
DeleteObject(hBrush);
/* delete green brush */
InvalidateRect(hWnd, NULL, 1);
break;
        case IDM_SIZE:
          X += 10;
          Y += 10;
          InvalidateRect(hWnd, NULL, 1);
          break;
        case IDM_ORG:
          orgX += 50;
          orgY += 50;
          InvalidateRect(hWnd, NULL, 1);
          break;
        case IDM_RESET:
          orgX = 0;
          orgY = 0;
          X = 40;
          Y = 40;
        case IDM_SHOW:
          ShowWindow(tbWnd, SW_RESTORE);
          break;
        case IDM_HIDE:
          ShowWindow(tbWnd, SW_HIDE);
          break;
        case IDM_CLEAR:
       /* reset current position to 0,0 */
          MoveToEx(memdc, 0, 0, NULL);

      /* erase by repainting background */
          PatBlt(memdc, 0, 0, maxX, maxY,
                PATCOPY);
          InvalidateRect(hWnd, NULL, 1);
          break;
```

```
                case IDM_ABOUT :
                  /* Show abour button as pressed*/
                  SendMessage(tbWnd, TB_CHECKBUTTON, (LPARAM)
                              IDM_ABOUT, (WPARAM) 1);
                  DialogBox( hInst, "AboutBox", hWnd,
                              (DLGPROC)About );

                  /* Reset the help button */
                  SendMessage(tbWnd, TB_CHECKBUTTON, (LPARAM)
                              IDM_ABOUT, MB_OK);
                  break;

                case IDM_EXIT :
                  DestroyWindow( hWnd );
                  break;
              }
              break;
      case WM_PAINT: /* process a repaint request */
            hDC = BeginPaint(hWnd, &paintstruct); /* get DC */

            /* set mapping mode, window and viewport extents */
            SetMapMode(hDC, MM_ANISOTROPIC);
            SetWindowExtEx(hDC, 200, 200, NULL);
            SetViewportExtEx(hDC, X, Y, NULL);
            SetViewportOrgEx(hDC, orgX, orgY, NULL);

            /* now, copy memory image onto screen */
            BitBlt(hDC, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
            EndPaint(hWnd, &paintstruct); /* release DC */
            break;
      case WM_DESTROY :
            DeleteObject(hRedpen);
            DeleteObject(hGreenpen);
            DeleteObject(hBluepen);
            DeleteObject(hYellowpen);

            DeleteDC(memdc); /* delete the memory device */
            PostQuitMessage(0);
            break;
      default :
            return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
   }

   return( 0L );
}
```

```
LRESULT CALLBACK About( HWND hDlg,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
   switch (message)
   {
       case WM_INITDIALOG:
               return (TRUE);

       case WM_COMMAND:
               if (    LOWORD(wParam) == IDOK
                    || LOWORD(wParam) == IDCANCEL)
               {
                       EndDialog(hDlg, TRUE);
                       return (TRUE);
               }
               break;
   }

   return (FALSE);
}

/* Initialize the toolbar structures. */
  void InitToolbar()
{
  tbButtons[0].iBitmap = 0;
  tbButtons[0].idCommand = IDM_LINES;
  tbButtons[0].fsState = TBSTATE_ENABLED;
  tbButtons[0].fsStyle = TBSTYLE_BUTTON;
  tbButtons[0].dwData = 0L;
  tbButtons[0].iBitmap = 0;
  tbButtons[0].iString = 0;

  tbButtons[1].iBitmap = 1;
  tbButtons[1].idCommand = IDM_RECTANGLES;
  tbButtons[1].fsState = TBSTATE_ENABLED;
  tbButtons[1].fsStyle = TBSTYLE_BUTTON;
  tbButtons[1].dwData = 0L;
  tbButtons[1].iString = 0;
```

```
tbButtons[2].iBitmap = 2;
tbButtons[2].idCommand = IDM_ELLIPSES;
tbButtons[2].fsState = TBSTATE_ENABLED;
tbButtons[2].fsStyle = TBSTYLE_BUTTON;
tbButtons[2].dwData = 0L;
tbButtons[2].iString = 0;

/* button separator */
tbButtons[3].iBitmap = 0;
tbButtons[3].idCommand = 0;
tbButtons[3].fsState = TBSTATE_ENABLED;
tbButtons[3].fsStyle = TBSTYLE_SEP;
tbButtons[3].dwData = 0L;
tbButtons[3].iString = 0;

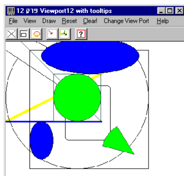
tbButtons[4].iBitmap = 3;
tbButtons[4].idCommand = IDM_SIZE;
tbButtons[4].fsState = TBSTATE_ENABLED;
tbButtons[4].fsStyle = TBSTYLE_BUTTON;
tbButtons[4].dwData = 0L;
tbButtons[4].iString = 0;

tbButtons[5].iBitmap = 4;
tbButtons[5].idCommand = IDM_ORG;
tbButtons[5].fsState = TBSTATE_ENABLED;
tbButtons[5].fsStyle = TBSTYLE_BUTTON;
tbButtons[5].dwData = 0L;
tbButtons[5].iString = 0;

tbButtons[6].iBitmap = 0;
tbButtons[6].idCommand = 0;
tbButtons[6].fsState = TBSTATE_ENABLED;
tbButtons[6].fsStyle = TBSTYLE_SEP;
tbButtons[6].dwData = 0L;
tbButtons[6].iString = 0;

tbButtons[7].iBitmap = 5;
tbButtons[7].idCommand = IDM_ABOUT;
tbButtons[7].fsState = TBSTATE_ENABLED;
```

```
  tbButtons[7].fsStyle = TBSTYLE_BUTTON;
  tbButtons[7].dwData = 0L;
  tbButtons[7].iString = 0;
}
```



תרשים 12.3: פלט דוגמה של תוכנית סרגל הכלים בשלמותה, כולל תוויות הלחצן

בפרק הבא תמשיך לחקור את הפקדים המשותפים, **פקדי מעלה-מטה** (שנקראים גם פקדי Spin), **פסי עקיבה** (Track Bars) **ופסי התקדמות** (Progress Bars).

פרק **13**

פקדים משותפים נוספים

בפרק זה תמשיך לסקור את הפקדים המשותפים של חלונות 9x. כאשר תתמקד בפקד מעלה-מטה (Up-Down), פס העקיבה (Trackbar) ופס ההתקדמות (Progress Bar).

13.1 פקדי מעלה-מטה

אחד הפקדים השימושיים ביותר הוא פקד **מעלה-מטה** (Up-Down Control). פקד זה הוא, למעשה, **פס גלילה** (Scroll Bar) ללא הפס! הוא מכיל רק את החיצים שבקצות פס הגלילה, אך אינו מכיל את הפס שביניהם. בוודאי הבחנת כי קיימים יישומי חלונות שפסי הגלילה שלהם קטנים ולמעשה, חסרי משמעות. קיימים גם מצבים שאינם מתאימים לעקרון של פס גלילה, אך מפיקים תועלת מהשימוש בחיצים. כדי לתת מענה למצבים כדוגמת אלה, יצרו מפתחי חלונות 9x את הפקד מעלה-מטה, הדומה מאוד לפס הגלילה.

ניתן להפעיל פקד מעלה-מטה באחת משתי דרכים: כפס גלילה עצמאי, ובשילוב עם פקד נוסף, שנקרא **חלון חבר** (Buddy Window). חלון החבר הנפוץ ביותר הוא **תיבת עריכה** (Edit Box). במקרה זה, נוצר פקד spin (או Spinner). כשמשתמשים בפקד spin, המערכת מספקת את כל התקורה הדרושה לטיפול בפקד באופן אוטומטי, ולכן קל מאוד להוסיפו ליישומים. בפרק זה נציג שתי דוגמאות לפקדי מעלה-מטה. בדוגמה הראשונה ניצור פקד עצמאי, ואילו בשנייה נשלב את הפקד עם חלון חבר, כדי לקבל פקד spin. להזכירך, פקד spin הוא למעשה, פקד מעלה-מטה מקושר לתיבת עריכה.

יצירת פקד מעלה-מטה

ליצירת פקד מעלה-מטה, הפעל את הפונקציה ()CreateUpDownControl המתוארת להלן:

```
HWND CreateUpDownControl(DWORD Style, int X, int Y, int Width,
                         int Height, HWND hParent, int ID,
                         HINSTANCE hInst, HWND hBuddy, int Max,
                         int Min, int StartPos);
```

כאן, הפרמטר Style מציין את סגנון הפקד. פרמטר זה חייב להכיל את הסגנונות WS_CHILD, WS_VISIBLE ו-WS_BORDER, ויכול להכיל גם סגנון אחד, או יותר מאלה המופיעים בטבלה 13.1.

המשתנים X ו-Y מעבירים את הקואורדינטות של הפינה השמאלית-עליונה של הפקד. Width ו-Height מציינים את רוחב הפקד וגבהו.

hParent מעביר את הידית של חלון האב. המספר המזהה הקשור לפקד מוגדר על ידי הפרמטר ID. hInst מעביר את ידית המופע של היישום. hBuddy מעביר את ידית החלון החבר. אם לא קיים חלון חבר, פרמטר זה חייב לקבל ערך NULL.

טווח הפעולה של הפקד מועבר על ידי הפרמטרים Max ו-Min. אם Max קטן מ-Min, הפקד ינוע בכיוון ההפוך. המיקום ההתחלתי של הפקד (חייב להיות בתחום שהוגדר עבורו) מועבר על ידי StartPos.

הפונקציה מחזירה ידית אל הפקד.

טבלה 13.1: הסגנונות של פקד מעלה-מטה

סגנון	משמעות
UDS_ALIGNLEFT	מיישר את הפקד לצידו השמאלי של החלון החבר.
UDS_ALIGNLEFT	מיישר את הפקד לצידו הימני של החלון החבר.
UDS_ARROWKEYS	מאפשר הפעלה של מקשי החיצים (כלומר, מאפשר להזיז את הפקד בעזרת מקשי החיצים).
UDS_AUTOBUDDY	החלון החבר הוא החלון הקודם בסדר החלונות.
UDS_HORZ	פקד מעלה-מטה הוא אופקי (תצורת המחדל של פקדים אלה היא מאונכת).
UDS_NOTHOUSANDS	אין שימוש בפסיקי הפרדה בערכים גדולים (ישים לפקדי spin בלבד).
UDS_SETBUDDYINT	מארגן אוטומטית את הטקסט בחלון החבר כשמשתנה מיקום הפקד, מאפשר לחלון החבר להציג את המיקום הנוכחי של הפקד.
UDS_WRAP	מיקום הפקד יחזור להתחלה בעת ניסיון להזיזו מעבר לנקודת הסיום.

הודעות פקד מעלה-מטה

כשלוחצים על אחד מחיצי הפקד, הוא שולח הודעת WM_VSCROLL אל חלון האב שלו. lParam יכיל את הידית של הפקד. הודעות WM_VSCROLL עשויות להישלח על ידי פקדים שונים, ולכן יש לבדוק אם ההודעה שב-lParam נוצרה על ידי פקד מעלה-מטה. לקבלת המיקום החדש, שלח הודעת UDM_GETPOS אל הפקד (ניתן לשלוח את הודעות הבקרה באמצעות ()SendMessage). הפונקציה מחזירה את המיקום הנוכחי של הפקד.

בנוסף להודעת UDM_GETPOS, פקדי מעלה-מטה מגיבים להודעות נוספות. טבלה 13.2 מציגה את ההודעות הנפוצות ביותר. לדוגמה, התוכנית יכולה לקבוע את מיקום הפקד בעזרת הודעת UDM_SETPOS.

טבלה 13.2: הודעות מעלה-מטה מקובלות

הודעה	משמעות
UDM_GETBUDDY	מקבלת את הידית של חלון החבר, שנמצאת בחלק הנמוך במילה של הערך המוחזר. WParam הוא 0. LParam הוא 0.
UDM_GETPOS	מקבלת את המיקום הנוכחי, שנמצא בחלק הנמוך במילה של הערך המוחזר. WParam הוא 0. LParam הוא 0.
UDM_GETRANGE	מקבלת את התחום הנוכחי. הערך המקסימלי נמצא בחלק הנמוך במילה של הערך המוחזר, והערך המינימלי נמצא בחלק הגבוה במילה של הערך המוחזר. WParam הוא 0. LParam הוא 0.
UDM_SETBUDDY	מגדירה חלון חבר חדש. מחזירה את הידית של החלון החבר הישן. WParam הוא הידית של החלון החבר החדש. LParam הוא 0.
UDM_SETPOS	קובעת את המיקום נוכחי. WParam הוא 0. LParam הוא המיקום הנוכחי החדש.
UDM_SETRANGE	קובעת את התחום הנוכחי. WParam הוא 0. LParam הוא החלק הנמוך של המילה מכיל את הערך המקסימלי; החלק הגבוה של המילה מכיל את הערך המינימלי.

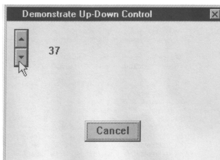
הפעלת פקד מעלה מטה

התוכנית **UpDown** הבאה יוצרת פקד מעלה-מטה עצמאי בתוך תיבת דו-שיח. בדוגמה זו הפקד אינו מקושר לחלון החבר. תחום הפקד הוא 0-100, ומיקומו ההתחלתי 50. כל פעם שמשנים את מיקום הפקד (באמצעות לחיצה על החץ), מוצג המיקום החדש באזור הלקוח של תיבת הדו-שיח.

התוכנית בשלמותה נמצאת בתקליטור Chap13\UpDown . (הבסיס לתוכנית זו היא DlgBox).

תיבת הדו-שיח בתוכנית מכילה את הפקד מעלה-מטה. תרשים 13.1 מציג דוגמת פלט של התוכנית. הפקד נוצר כאשר מאתחלים את תיבת הדו-שיח באמצעות הקוד הבא:

```
case WM_INITDIALOG:
      udWnd = CreateUpDownControl(
                  WS_CHILD | WS_BORDER | WS_VISIBLE,
                  10, 10, 50, 50,
                  hDlg,
                  ID_UPDOWN,
                  hInst,
                  NULL,
                  100, 0, 50);
      return 1;
```



תרשים 13.1: פלט של תוכנית הדוגמה הראשונה של פקד מעלה-מטה

הקריאה לפונקציה ()CreateUpDownControl יוצרת פקד מעלה-מטה במיקום 10,10 בתוך תיבת הדו-שיח. גודל הפקד בפיקסלים הוא 50x50. הפקד הוא חלון-בן של תיבת הדו-שיח, ולכן הידית שלה (hdwnd) מועברת כידית אב. המספר המזהה של הפקד הוא ID_UPDOWN. הדוגמה שלפנינו היא פשוטה ואינה זקוקה לערך זה, אך תוכניות אחרות עשויות להזדקק לו. hInst היא ידית המופע של התוכנית. הפרמטר החבר מקבל ערך NULL כיון שאין כאן שימוש בחלון חבר. תחום הפעולה של הפקד הוא 0 עד 100, והמיקום ההתחלתי הוא 50.

כל פעם שניגשים אל הפקד, נשלחת הודעת WM_VSROLL אל תיבת הדו-שיח. הקוד שמטפל בהודעה יוצג בהמשך. lParam מכיל את הידית של הפקד. הידית נבדקת ומוחזרת על ידי הפונקציה ()CreateUpDownCntrol, במטרה לוודא את זהות הפקד שיצר את ההודעה. הדוגמה שלפנינו כוללת פקד אחד בלבד, ולכן הפעולה נראית מיותרת. לעומת זאת, יישומים מעשיים עשויים להכיל פקדים רבים, שכל אחד מהם מסוגל ליצור הודעה, ולכן עליך להקפיד לבדוק את זהות הפקד.

```
case WM_VSCROLL:  /* manually process an up-down control */
      if(udWnd==(HWND)lParam) {
         udpos = SendMessage(udWnd, UDM_GETPOS, 0, 0);
         wsprintf(szBuffer, "%d", LOWORD(udpos));
         HDC hDC = GetDC( hDlg );
         TextOut(hDC, 55, 30, "       ", 6);
         TextOut(hDC, 55, 30, szBuffer, strlen(szBuffer));
         ReleaseDC(hDlg, hDC);
         return 1;
      }
```

לקבלת המיקום החדש של פקד מעלה-מטה שולחים הודעת UDM_GETPOS באמצעות ()SendMessage. המיקום נמצא בחלק הנמוך של הערך המוחזר, והערך מוצג באזור הלקוח של תיבת הדו-שיח.

13.2 יצירת פקד Spin

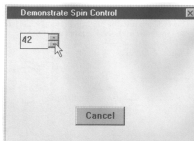
אין רע ביצירה של פקד מעלה-מטה עצמאי ובשימוש בו, אך בדרך כלל הוא קשור לתיבות עריכה. כשפקד מסוג זה משולב עם תיבת עריכה, נקרא פקד spin. חלונות 9x עושה שימוש נרחב בפקד spin, והיא מספקת אמצעי תמיכה מיוחדים עבורו. למעשה, פקד spin הוא אוטומטי לחלוטין ואין צורך לייעד לו משאבים כלשהם מתוך התוכנית.

כדי ליצור פקד spin, עליך להגדיר פקד עריכה כחלון חבר של פקד מעלה-מטה. לאחר מכן, יוצג המיקום החדש של הפקד בתיבת העריכה בכל פעם שיחול שינוי כלשהו בפקד. בנוסף, אם תשנה ידנית את הערך שבתיבת העריכה, יתבטא הדבר מייד בפקד.

תוכנית לדוגמה של פקד Spin

יצירת פקד spin היא תהליך פשוט המתבצע בשני שלבים. ראשית, הוספת תיבת עריכה לקובץ המשאבים של התוכנית. שנית, העברת ידית התיבה כחלון חבר בעת יצירת פקד מעלה-מטה. התוכנית בשלמותה נמצאת בתקליטור Chap13\Spin .

תרשים 13.2 מציג דוגמת פלט של התוכנית.



תרשים 13.2: דוגמת פלט של תוכנית ההדגמה של פקד spin

כדי להבין כיצד נוצר פקד spin, עיין בקוד של case WM_INITDIALOG להלן:

```
case WM_INITDIALOG:
  hEboxWnd = GetDlgItem(hdwnd, ID_EB1);
  udWnd = CreateUpDownControl(
                WS_CHILD | WS_BORDER | WS_VISIBLE |
                UDS_SETBUDDYINT | UDS_ALIGNRIGHT,
                10, 10, 50, 50,
                hdwnd,
                ID_UPDOWN,
                hInst,
                hEboxWnd,
                100, 0, 50);
  return 1;
```

תיבת הדו-שיח מוגדרת בקובץ המשאבים, ולכן התוכנית חייבת לקרוא לפונקציה GetDlgItem() כדי לקבל את הידיות שלה. להלן ההגדרה של GetDlgItem():

```
HWND GetDlgItem(HWND hDialog, int ID);
```

הפונקציה GetDlgItem מחזירה ידית אל הפקד שנבחר. הידית של תיבת הדו-שיח שמכילה את הפקד מוגדרת ב-hDialog. ID מעביר את המספר המזהה של הפקד. במקרה של שגיאה, מחזירה הפונקציה ערך NULL.

ברגע שהתקבלה הידית של תיבת הדו-שיח, היא מועברת כחלון חבר אל הפונקציה CreateUpDownControl(). ברגע שנוצר הפקד בשילוב עם תיבת העריכה בתור החלון החבר שלו, חל קישור אוטומטי של השניים, ונוצר פקד spin.

13.3 פס העקיבה

אחד הפקדים המצודדים ביותר מבחינה חזותית, הוא **פס העקיבה** (Trackbar), שנקרא לעיתים גם **פקד גררה** (Slider). פקד זה מזכיר בצורתו את הגררה המצויה בסוגי ציוד אלקטרוני רבים, כגון מערכות סטריאו. פקד זה מכיל מתג שנע בתוך מסילה. התוכנית מטפלת בו באופן דומה, למרות שמראהו שונה לחלוטין מפס הגלילה.

פסי עקיבה שימושיים במיוחד כאשר שהתוכנית מיועדת לפיקוח על התקן אמיתי. לדוגמה, פסי עקיבה הם אמצעי מעולה לשליטה באקווליייזר גרפי וקביעת עקומת התדרים שלו.

ליצירת פס עקיבה, השתמש בפונקציה CreateWindow(), או בפונקציה CreateWindowEx(). האחרונה מספקת מפרט מוגדל יותר של סגנונות. אין צורך במפרט הסגנונות המוגדל עבור פס העקיבה בדוגמה שתוצג בפרק, אך ייתכן שתמצא לו שימוש ביישומים שתכתוב. **מחלקת החלון** (Window Class) של פס העקיבה הוא TRACKBAR_CLASS, ולמעשה זהו העיצוב של החלון.

סגנונות של פסי עקיבה

מגוון סגנונות עומד לרשותך ביצירת פס עקיבה. הנפוצים שבהם מופיעים בטבלה 13.3. ברוב המקרים תכלול את הסגנון TBS_AUTOTICKS, מכיון שהוא מציג סימני מידה זעירים לצד מסילת הגררה. סימנים אלה הם הסרגל (סקאלה) של פס העקיבה.

טבלה 13.3: אפשרויות הסגנון של פס העקיבה

סגנון	האפקט החזותי שלו
TBS_AUTOTICKS	סימני מידה לצד מסילת פס העקיבה
TBS_HORZ	פס עקיבה אופקי (ברירת המחדל)
TBS_VERT	פס עקיבה אנכי
TBS_BOTTOM	סימני מידה בתחתית הפס (ברירת המחדל)
TBS_TOP	סימני מידה מעל הפס
TBS_LEFT	סימני מידה משמאל לפס
TBS_RIGHT	סימני מידה מימין לפס
TBS_BOTH	סימני מידה משני צידי לפס

משלוח הודעות פס העקיבה

בדומה לפקדים משותפים אחרים שפגשת, גם כאן מעבירים הודעות אל פס העקיבה באמצעות הפונקציה ()SendMessage. טבלה 13.4 מציגה את ההודעות הנפוצות של פס העקיבה. קיימות שתי הודעות שיהיה עליך לשלוח כמעט תמיד: TBM_SETRANGE ו-TBM_SETPOS. הודעות אלו קובעות את תחום הפעולה של הפקד ואת מיקומו ההתחלתי, בהתאמה. לא ניתן לקבוע מאפיינים אלה בעת יצירת הפקד בעזרת ()CreateWindow.

טבלה 13.4: הודעות אופייניות של פס העקיבה

הודעה	משמעות
TBM_GETPOS	קבלת המיקום הנוכחי. wParam הוא 0. lParam הוא 0.
TBM_GETRANGEMAX	קבלת התחום המקסימלי של פס העקיבה. wParam הוא 0. lParam הוא 0.
TBM_GETRANGEMIN	קבלת התחום המינימלי של פס העקיבה. wParam הוא 0. lParam הוא 0.
TBM_SETPOS	קביעת המיקום הנוכחי. wParam מקבל ערך שונה מ-0 לצורך ציור הפקד מחדש, או 0 בכל מקרה אחר. lParam מכיל את המיקום החדש.
TBM_SETRANGE	קביעת תחום הפעולה של הפקד. wParam מקבל ערך שונה מ-0 לצורך ציור הפקד מחדש, או 0 בכל מקרה אחר. lParam מכיל את תחום הפעולה. קצהו התחתון של התחום נמצא בחלק הנמוך של המילה, ואילו חלקה הגבוה מכיל את קצהו העליון.
TBM_SETRANGEMAX	קביעת התחום המקסימלי. wParam מקבל ערך שונה מ-0 לצורך ציור הפקד מחדש, או 0 בכל מקרה אחר. lParam מכיל את הערך המקסימלי של תחום הפעולה.
TBM_SETRANGEMIN	קביעת התחום המינימלי. wParam מקבל ערך שונה מ-0 לצורך ציור הפקד מחדש, או 0 בכל מקרה אחר. lParam מכיל את הערך המקסימלי של תחום הפעולה. lParam מכיל את הערך המינימלי של תחום הפעולה.

טיפול בהודעת פס עקיבה

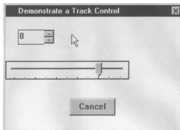
כשפונים לפס העקיבה, נוצרת הודעת הגלילה WM_HSCROLL. החלק הנמוך של המילה wParam מכיל ערך שמייצג את מהות הפעולה. lParam מכיל את הידית של פס העקיבה שיצר את ההודעה. טבלה 13.5 מציגה חלק מההודעות המקובלות של פס העקיבה.

טבלה 13.5: הודעות notification אופייניות של פס העקיבה

הודעה	משמעות
TB_BOTTOM	נלחץ מקש END ; הגררה הגיעה עד לערך המינימלי
TB_ENDTRACK	סיום פעולת פס העקיבה
TB_LINEDOWN	נלחץ מקש חץ ימינה או חץ למטה
TB_LINEUP	נלחץ מקש חץ שמאלה או חץ למעלה
TB_PAGEDOWN	נלחץ מקש PGDN, או לחיצה בעכבר לפני הגררה
TB_PAGEUP	נלחץ מקש PGUP, או לחיצה בעכבר אחרי הגררה
TB_THUMBPOSITION	הגררה הוזזה בעזרת העכבר
TB_THUMBTRACK	הגררה נגררה בעזרת העכבר
TB_POP	נלחץ מקש HOME

תוכנית הדגמה של פס העקיבה

התוכנית **All13** הבאה מוסיפה פס עקיבה לתוכנית הקודמת של פקד spin. כשתריץ אותה, תיווכח לגלות כי כל שינוי בפס העקיבה גורר שינוי מקביל בפקד spin, ולהיפך. באופן זה ניתן להדגים שליחת הודעות וקבלתן על ידי פס העקיבה. תרשים 13.3 מציג דוגמת פלט של התוכנית. התוכנית בשלמותה נמצאת בתקליטור Chap13\All13.



תרשים 13.3: דוגמת פלט של תוכנית פס העקיבה

עם הצגת תיבת הדו-שיח נוצרים הפקדים spin ופס העקיבה. תחום הפעולה של פס העקיבה נקבע ל-0 עד 10 בעת יצירתו, והגררה מוצבת על הערך 5 (התחום והערך ההתחלתי האלה ניתנים גם לפקד spin). שים לב שהתחום נקבע בעזרת המאקרו ()MAKELONG, אשר מצרף שני ערכים שלמים (Integers) לערך שלם ארוך. זוהי תבנית הפקודה:

```
DWORD MAKELONG(WORD low, WORD high);
```

הפרמטר low מכיל את החלק הנמוך של המילה הכפולה של הערך הארוך, והפרמטר high מכיל את חלקה הגבוה. המאקרו ()MAKELONG שימושי מאוד במקרים שיש צורך לקודד שתי מילים לערך שלם ארוך.

כל שינוי בפקד spin גורר הודעת WM_VSCROLL ושינוי מיקומו של פס העקיבה בהתאם:

```
case WM_VSCROLL: /* process up-down control */
     if(udWnd==(HWND)lParam) {
        trackpos = GetDlgItemInt(hDlg, ID_EB1, NULL, 1);
        SendMessage(hTrackWnd, TBM_SETPOS, (WPARAM) 1,
                                   (LPARAM) trackpos);
                          }
     return 1;
```

משפט case שתואר מקבל את הערך החדש מתיבת העריכה באמצעות קריאה לפונקציה ()GetDlgItemInt. הפונקציה זהה לפונקציה ()GetDlgItemText, שלמדת בפרק 5, למעט העובדה שהיא מחזירה את הערך השלם השקול לתוכן בתיבה, ולא טקסט. לדוגמה, אם התיבה מכילה את המחרוזת 102, ()GetDlgItemInt תחזיר את הערך 102. מובן שהפונקציה ישימה לתיבות עריכה שמכילות ערכים מספריים בלבד. זוהי התבנית שלה:

```
UINT GetDlgItemInt(HWND hDialog, int ID,
                              BOOL *error, BOOL signed);
```

hDialog מעביר את ידית האחיזה של תיבת הדו-שיח שמכילה את פקד העריכה. ID מעביר את המספר המזהה של תיבת הדו-שיח. אם תיבת העריכה אינה מכילה ערך מספרי חוקי, תחזיר הפונקציה את הערך 0, שהוא ערך חוקי לכל דבר. לכן, הצלחת הפונקציה, או כשלונה מובעים על ידי משתנה שהמצביע אליו נמצא במשתנה error. משתנה זה יקבל ערך שונה מ-0 במקרה שהערך המוחזר הוא חוקי, ו-0 במקרה של שגיאה. אם תעדיף להתעלם משגיאות מסוג זה, תוכל להגדיר NULL במקום הפרמטר בפונקציה. הפונקציה תחזיר ערך עם **סימן** (Signed) במקרה שהפרמטר signed שונה מ-0, או ערך ללא סימן בכל מקרה אחר.

לאחר שנשלמו ההגדרות בתיבת העריכה, הן מועברות לפס העקיבה באמצעות הפונקציה ()SendMessage. כך, כל שינוי בערך פקד spin יתבטא באופן אוטומטי בפס העקיבה.

כל תנועה של פס העקיבה גורמת לקבלת הודעת WM_HSCROLL, שמטופלת כך :

```
case WM_HSCROLL: /* trackbar was activated */
      if(hTrackWnd != (HWND)lParam) break; /* not trackbar */

      switch(LOWORD(wParam)) {
         case TB_TOP:
         case TB_BOTTOM:         /* For this example */
         case TB_LINEDOWN:       /* all messages will be */
         case TB_LINEUP:         /* processed in the same */
         case TB_THUMBPOSITION: /* way. */
         case TB_THUMBTRACK:
         case TB_PAGEUP:
         case TB_PAGEDOWN:
            trackpos = SendMessage(hTrackWnd, TBM_GETPOS,
                                   0, 0);
            SetDlgItemInt(hDlg, ID_EB1, trackpos, 1);
         return 1;
                          }
      break;
```

כשהמשתמש מזיז את הגררה של פס העקיבה, מתעדכן המיקום באופן אוטומטי, והתוכנית פטורה מלטפל בכך. התוכנית מקבלת את הערך החדש של פס העקיבה לאחר הזזתו, ומנצלת אותו לעדכון פקד spin. הערך שמכילה תיבת העריכה של פקד זה נקבע בעזרת הפונקציה ()SetDlgItemInt. הפונקציה מבצעת את הפעולה ההפוכה של הפונקציה ()GetDlgItemInt שתוארה לעיל. וכך היא נכתבת :

```
BOOL SetDlgItemInt(HWND hDialog, int ID,
                   UINT value, BOOL signed);
```

הפרמטר hDialog מעביר את ידית האחיזה של תיבת הדו-שיח, שמכילה את פקד העריכה. ID מעביר את המספר המזהה של תיבת הדו-שיח. value מכיל את הערך שיש להציב בתיבת העריכה. אם הפרמטר signed מכיל ערך שונה מ-0, ניתן להעביר מספרים שליליים. אחרת, מתייחסים לערך שהוא מכיל כמספר ללא סימן (Unsigned). במקרה של הצלחה מחזירה הפונקציה מספר שונה מ-0, ובמקרה של כישלון היא מחזירה 0.

בדוגמה, ניתן להזיז את פס העקיבה בעזרת העכבר, או המקלדת. למעשה, המספר הגדול של ההודעות מסוג _TB מיועד לתמוך בממשק המקלדת. נסה לבטל מספר הודעות ובדוק את התוצאות.

הזיקה בין פס העקיבה ופקד spin בתוכנית שהצגנו היא שרירותית לחלוטין, ונועדה לצורך המחשה בלבד. ניתן לשלב ביישומים פסי עקיבה שפועלים בצורה עצמאית.

13.4 פס התקדמות

פס ההתקדמות (Progress Bar) הוא אחד הפקדים הפשוטים מתוך הפקדים המשותפים החדשים. בוודאי ראית כבר פס כזה בפעולה. פס התקדמות הוא חלון צר וארוך שמציג את התקדמות הביצוע של משימה ממושכת. לדוגמה, פסי התקדמות מקובלים מאוד בתוכניות התקנה, תוכניות העברת קבצים ותוכניות מיון.

יוצרים את פס ההתקדמות בעזרת הפונקציה ()CreateWindow, או ()CreateWindowEX, בעזרת הגדרת מחלקת חלון PROGRESS_CLASS.

שיגור הודעות פס התקדמות

התוכנית מנצלת את הפונקציה התקנית ()SendMessage כדי לשלוח את הודעות פס ההתקדמות (פס ההתקדמות אינו יוצר הודעות כלשהן). בדרך כלל יישלחו הודעות לקביעת תחום הפס והצגת מהלך התקדמות המשימה. טבלה 13.6 מציגה אחדות מההודעות הנפוצות של פס ההתקדמות.

תחום המחדל של פס ההתקדמות הוא 0 עד 100, אך ניתן לתת לו כל ערך בין 0 ל-65,535. הצגת התקדמות המשימה תיעשה בדרך כלל באמצעות הודעת PBM_STEPIT. ההודעה גורמת לפס להתקדם במנות קבועות מראש, שנקראות **פסיעות** (Steps). גודל המחדל של פסיעה הוא 10, אך תוכל לשנותו לפי הצורך. הפס ילך ויתמלא ככל שתקדם את מיקומו. פס ההתקדמות משמש להצגת ההתקדמות בביצוע המשימה, ולכן במצב מלא הוא מייצג השלמת המשימה ב-100%.

תוכנית פשוטה של פס התקדמות

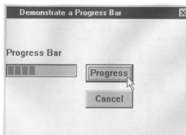
התוכנית **Progress** הבאה מדגימה כיצד משתמשים בפס ההתקדמות. התוכנית יוצרת תיבת דו-שיח שמכילה פס התקדמות ולחצן שנקרא Progress. תחום פס ההתקדמות הוא בין 0 ל-50, וגודל הפסיעה שלו 5. בכל פעם שלוחצים על הלחצן, מתקדם פס ההתקדמות פסיעה נוספת. ברגע שהפס מלא, נעלמת תיבת הדו-שיח מהמסך באופן אוטומטי.

טבלה 13.6: הודעות נפוצות של פס ההתקדמות

הודעה	משמעות
PBM_DELTAPOS	מקדם את פס ההתקדמות בפסיעה בגודל מסוים. מוחזר המיקום הקודם. wParam מכיל את גודל הפסיעה החדשה. lParam הוא 0.
PBM_SETPOS	קביעת המיקום של פס ההתקדמות. מוחזר המיקום הקודם. wParam מכיל את המיקום החדש. lParam הוא 0.

הודעה	משמעות
PBM_SETRANGE	קביעת התחום של פס ההתקדמות. התחום הקודם מוחזר כשערכו המקסימלי בחלק הגבוה של המילה, וערכו המינימלי בחלקה הנמוך. wParam הוא 0. lParam מכיל את התחום. ערכו המקסימלי בחלק הגבוה של המילה, וערכו המינימלי בחלקה הנמוך.
PBM_SETSTEP	קביעת מנת ההתקדמות (הפסיעה). מוחזר גודל הפסיעה הקודם. wParam מכיל את הפסיעה החדשה. lParam הוא 0.
PBM_STEPIT	קידום הפס בהתאם לגודל הפסיעה. wParam הוא 0. lParam הוא 0.

תרשים 13.4 מציג דוגמה של פלט התוכנית של פס ההתקדמות.



תרשים 13.4: דוגמה של פלט התוכנית של פס ההתקדמות

התוכנית בשלמותה נמצאת בתקליטור Chap13\Progress.

יש לזכור שפס ההתקדמות נועד לתת למשתמש ביטחון שהתוכנית מתקדמת באופן תקין. לכן אתה צריך לעדכן את פס ההתקדמות בתדירות גבוהה. זכור, המשתמש סומך על המידע שמציג פס ההתקדמות, שמעיד שהתוכנית עדיין פועלת. אם תשנה את הפס לאט מידי, משתמש קצר רוח עלול לאתחל את המחשב במחשבה שהתוכנית קרסה!

בפרק הבא תמשיך ותלמד על פקדים משותפים נוספים: **שורת המצב** (Status Bar), **פקד הכרטיסיה** (Tab Control) ו**פקד תצוגת העץ** (Tree View Control).

פרק 14

מבט נוסף על פקדים משותפים

בפרק זה תוצג סקירה נוספת על הפקדים המשותפים של חלונות 9x. נתמקד הפעם בחלון המצב (Status Window), פקד הכרטיסיה (Tab Control) ופקד תצוגת עץ (Tree View Control). לתשומת לבך, חלונות 9x כוללת פקדים משותפים נוספים שאותם תרצה בוודאי לחקור בעצמך.

14.1 חלון המצב

לעיתים יש צורך לדווח למשתמש על מצב משתנים, מאפיינים או פרמטרים בתוכנית. בעבר הגדירה כל תוכנית לעצמה את הדרך בה נעשה הדבר. חלונות 9x הנהיגה פקד תקני למטרה זו, שנקרא **חלון המצב** (Status Window) או **שורת המצב** (Status Bar). חלון המצב הוא שורה שמוצגת לרוחב תחתית המסך, וכוללת מידע הקשור לתוכנית.

תיווכח לדעת שקל ליישם את שורות המצב. שילוב שורת מצב ביישום כרכיב שגרתי מאפשר ממשק עקבי להצגת מצב היישום.

יצירת חלון מצב

ניתן ליצור שורת מצב בעזרת הפונקציה ()CreateStatusWindow. בנוסף, יש להביא בחשבון ששורות מצב **הן** חלונות, ולכן ניתן ליצור אותן בעזרת ()CreateWindow או ()CreateWindowEx, בציון מחלקת החלון STATUSCLASSNAME. כדי להמחיש זאת, ניצור שורת מצב בעזרת הפונקציה ()CreateWindow (תוכל, כמובן, לעשות זאת בכל דרך אחרת).

חלון מצב יהיה בדרך כלל חלון בן. יוצרים אותו בעזרת הסגנון WS_VISIBLE, ולכן הוא גם יוצג באופן אוטומטי. הקוד הבא יוצר חלון מצב:

```
hStatusWnd = CreateWindow(STATUSCLASSNAME,
               "", /* not used */
               WS_CHILD | WS_VISIBLE,
               0, 0, 0, 0, /* size and position ignored */
```

```
                hwnd, /* handle of parent */
                NULL,
                hInst, /* instance handle */
                NULL
              );
```

כפי שמשתמע מההערה בתוכנית, לא מעבירים את הפרמטרים של גודל החלון ומיקומו אל הפונקציה ()CreateWindow. חלון המצב מתאים את עצמו אוטומטית לחלון האב, ולכן אין צורך בפרמטרים אלה.

חלון המצב מתחלק בדרך כלל לחלקים (למרות שאין כל פסול בחלון שמכיל חלק אחד בלבד). ניתן לכתוב טקסט בכל אחד מחלקי החלון בנפרד, כאשר ההתייחסות לחלקים השונים נעשית באמצעות אינדקס (האינדקס של החלק הראשון הוא 0).

הודעות חלון המצב

חלון המצב אינו יוצר הודעות כלשהן, אך התוכנית יכולה לשלוח הודעות מצב באמצעות הפונקציות התקניות ()SendMessage. טבלה 14.1 מציגה את ההודעות הנפוצות של חלון המצב.

טבלה 14.1: ההודעות הנפוצות של שורת המצב

הודעה	משמעות
SB_GETPARTS	קבלת הקואורדינטה של צד ימין של כל אחד מחלקי החלון. מחזירה את מספר החלקים של שורת המצב. wParam מציין את מספר חלקי חלון המצב. lParam הוא מצביע אל מערך של מספרים שלמים (Int) שיקבל את הקואורדינטות של צידו הימני של כל חלק. גודל המערך צריך להיות לפחות כמספר החלקים בשורה.
SB_GETTEXT	קבלת הטקסט מהחלק שצויין. המילה הנמוכה בערך המוחזר מכילה את מספר התווים בטקסט. המילה הגבוהה מכילה ערך שמתאר כיצד להציג את הטקסט. אם היא מכילה את הערך 0, הטקסט יוצג בגובה נמוך מהחלון. אם היא שווה ל-SBT_POPOUT, הטקסט יוצג גבוה החלון. אם היא שווה ל-SBT_NOBORDERS, הטקסט יוצג ללא גבול (Border). wParam מציין את האינדקס של החלק המבוקש. lParam מצביע אל מערך תווים שמיועד לקבל את הטקסט (ודא שהמערך גדול במידה הדרושה ויכול להכיל את הטקסט).

הודעה	משמעות
SB_SETPARTS	ציון מספר החלקים בשורת המצב. מחזירה ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה של כישלון. wParam מציין את מספר החלקים. lParam הוא מצביע אל מערך של מספרים שלמים, שמכילים את הקואורדינטות של הצד הימני של כל אחד מחלקי השורה.
SB_GETTEXT	הצגת הטקסט בחלק השורה שנבחר. מחזיר ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה של כישלון. wParam מציין את האינדקס של חלק השורה אשר יקבל את הטקסט ואת אופן הצגתו. התוכנית מבצעת פעולת OR בין האינדקס וערך התצוגה. אם ערך התצוגה הוא 0, אזי הטקסט יופיע בגובה נמוך מהחלון (ברירת המחדל). אם ערך התצוגה הוא SBT_POPOUT, הטקסט יוצג גבוה מהחלון. אם הוא SBT_NOBORDERS, הטקסט יוצג ללא גבול (Border). אם ערך התצוגה הוא SBT_OWNERDRAW, יוצג חלון האב. lParam הוא מצביע אל מחרוזת המיועדת להצגה.

כמעט כל היישומים שולחים הודעות SB_SETPARTS שתפקידם לקבוע את מספר החלקים של שורת המצב, גם SB_SETTEXT שכותב טקסט בחלק מוגדר של חלון המצב. להלן התהליך הכללי ליצירת שורת המצב:

1. צור את חלון המצב.
2. קבע את מספר החלקים בחלון בעזרת הודעת SB_SETPARTS.
3. כתוב טקסט בכל אחד מהחלקים, בעזרת הודעת SB_SETTEXT.

מרגע ששורת המצב עברה את שלב האתחול, תוכל לעדכן את חלקיה השונים בהתאם לצורך, על ידי שיגור הודעת SB_SETTEXT.

הפעלת שורת המצב

התוכנית **Status_bar** הבאה (בתקליטור Chap14) מציגה את ההגדרות של תיבת הדו-שיח באמצעות שורת מצב. תיבת הדו-שיח מכילה פקד spin ושתי **תיבות סימון** (Check Boxes). בכל פעם שחל שינוי באחד הפקדים, מתעדכן הסטטוס שלו בשורת המצב.

להלן תוכנית שורת המצב (**Status_bar**). תרשים 14.1 מציג פלט לדוגמה של התוכנית.

```
#include <windows.h>
#include <commctrl.h>
#include "Status_bar.h"
#include <stdio.h>
#define IS_WIN32 TRUE

/* Demonstrate a status bar. */

#define NUMPARTS 3

HINSTANCE hInst;         // current instance

LPCTSTR lpszAppName = "StatusBar";
LPCTSTR lpszTitle =   "Using a Status Bar";

BOOL RegisterWin95(CONST WNDCLASS* lpwc);
int parts[NUMPARTS];

extern void InitStatus(HWND hWnd);
HWND hStatusWnd;

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE
                   hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
 {
   MSG msg;
   HWND hWnd;
   WNDCLASS wc;

   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = (WNDPROC)WndProc;
   wc.cbClsExtra     = 0;
   wc.cbWndExtra     = 0;
   wc.hInstance      = 0;
   wc.hIcon          = LoadIcon(hInstance, lpszAppName);
   wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName   = lpszAppName;
   wc.lpszClassName  = lpszAppName;
```

```
   if(!RegisterWin95(&wc))
      return false;
   hInst = hInstance;
   hWnd = CreateWindow (lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                       );
   if(!hWnd)
      return false;
   InitCommonControls();
   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
   while(GetMessage(&msg, NULL, 0,0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
   return(msg.wParam);
 }

BOOL RegisterWin95(CONST WNDCLASS* lpwc)
 {
   WNDCLASSEX wcex;

   wcex.style           = lpwc->style;
   wcex.lpfnWndProc     = lpwc->lpfnWndProc;
   wcex.cbClsExtra      = lpwc->cbClsExtra;
   wcex.cbWndExtra      = lpwc->cbWndExtra;
   wcex.hInstance       = lpwc->hInstance;
   wcex.hIcon           = lpwc->hIcon;
   wcex.hCursor         = lpwc->hCursor;
   wcex.hbrBackground   = lpwc->hbrBackground;
   wcex.lpszMenuName    = lpwc->lpszMenuName;
```

```
   wcex.lpszClassName   = lpwc->lpszClassName;
   wcex.cbSize          = sizeof(WNDCLASSEX);
   wcex.hIconSm         = LoadIcon(wcex.hInstance, "SMALL");
   return RegisterClassEx(&wcex);
 }

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam)
 {
   switch(uMsg)
    {
      case WM_COMMAND:
         switch(LOWORD(wParam))
          {
            case IDM_TEST :
               DialogBox(hInst, "STATUSDLG", hWnd,
                         (DLGPROC)DialogFunc);
               break;
            case IDM_EXIT :
               DestroyWindow(hWnd);
               break;
          }
         break;
      case WM_DESTROY :
         PostQuitMessage(0);
         break;
      default:
         return (DefWindowProc(hWnd, uMsg, wParam, lParam));
    }
   return(0L);
 }

/* A simple dialog function. */
LRESULT CALLBACK DialogFunc(HWND hDlg, UINT message,
                            WPARAM wParam, LPARAM lParam)
{
  static long udpos = 0;
  static char str[80];
  static HWND hEboxWnd;
  static HWND udWnd;
  static statusCB1, statusCB2;
  int low=0, high=20;
```

```
  switch(message) {
    case WM_INITDIALOG:
     InitStatus(hDlg); /* initialize the status bar */
     hEboxWnd = GetDlgItem(hDlg, ID_EB1);
     udWnd = CreateUpDownControl(
                   WS_CHILD | WS_BORDER | WS_VISIBLE |
                   UDS_SETBUDDYINT | UDS_ALIGNRIGHT,
                   10, 10, 50, 50,
                   hDlg,
                   ID_UPDOWN,
                   hInst,
                   hEboxWnd,
                   high, low, high/2);
return (TRUE);
    case WM_VSCROLL: /* process up-down control */
      if(udWnd==(HWND)lParam) {
        udpos = GetDlgItemInt(hDlg, ID_EB1, NULL, 1);
        sprintf(str, "Up-down: %d", udpos);
        SendMessage(hStatusWnd, SB_SETTEXT,
                    (WPARAM) 0, (LPARAM) str);
      }
      return (TRUE);
    case WM_COMMAND:
      switch(LOWORD(wParam)) {
        case ID_CB1: /* process checkbox 1 */
          statusCB1 = SendDlgItemMessage(hDlg, ID_CB1,
                    BM_GETCHECK, 0, 0);
          if(statusCB1) sprintf(str, "Option 1 ON");
          else sprintf(str, "Option 1 OFF");
          SendMessage(hStatusWnd, SB_SETTEXT,
                    (WPARAM) 1, (LPARAM) str);
          return (TRUE);
       case ID_CB2: /* process checkbox 2 */
          statusCB2 = SendDlgItemMessage(hDlg, ID_CB2,
                    BM_GETCHECK, 0, 0);
          if(statusCB2) sprintf(str, "Option 2 ON");
          else sprintf(str, "Option 2 OFF");
          SendMessage(hStatusWnd, SB_SETTEXT,
                    (WPARAM) 2, (LPARAM) str);
          return (TRUE);
```

```
        case ID_RESET: /* reset options */
          SendMessage(udWnd, UDM_SETPOS, 0, (LPARAM) high /2);
          SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 0,
                      (LPARAM) "Up-down: 10");
          SendDlgItemMessage(hDlg, ID_CB1,
                      BM_SETCHECK, 0, 0);
          SendDlgItemMessage(hDlg, ID_CB2,
                      BM_SETCHECK, 0, 0);
          SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 1,
                      (LPARAM) "Option 1: OFF");
          SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 2,
                      (LPARAM) "Option 2: OFF");
          return (TRUE);
        case IDCANCEL:
        case IDOK:
          EndDialog(hDlg, 0);
          return (TRUE);
      }
  }
  return 0;
}
/* Initialize the status bar. */
void InitStatus(HWND hWnd)
{
  RECT WinDim;
  int i;

  GetClientRect(hWnd, &WinDim);

  for(i=1; i<=NUMPARTS; i++)
    parts[i-1] = WinDim.right/NUMPARTS * i;

  /* Create a status bar */
  hStatusWnd = CreateWindow(STATUSCLASSNAME,
              "", /* not used in this example */
              WS_CHILD | WS_VISIBLE,
              0, 0, 0, 0,
              hWnd,
              NULL,
              hInst,
              NULL
      );
```

```
  SendMessage(hStatusWnd, SB_SETPARTS,
              (WPARAM) NUMPARTS, (LPARAM) parts);
  SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 0,
              (LPARAM) "Up-down: 10");
  SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 1,
              (LPARAM) "Option 1: OFF");
  SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 2,
              (LPARAM) "Option 2: OFF");
}
```

לתוכנית דרוש קובץ המשאבים הבא:

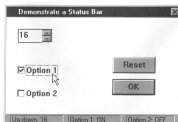
```
#include <windows.h>
#include "status.h"

MYMENU MENU
{
  MENUITEM "&Dialog", IDM_DIALOG
  MENUITEM "&Help", IDM_HELP
}

MYMENU ACCELERATORS
{
  VK_F2, IDM_DIALOG, VIRTKEY
  VK_F1, IDM_HELP, VIRTKEY
}

MYDB DIALOG 18, 18, 150, 92
CAPTION "Demonstrate a Status Bar"
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
{
  PUSHBUTTON "Reset", ID_RESET, 92, 34, 37, 14,
             WS_CHILD | WS_VISIBLE | WS_TABSTOP
  PUSHBUTTON "OK", IDOK, 92, 53, 37, 14,
             WS_CHILD | WS_VISIBLE | WS_TABSTOP
  EDITTEXT ID_EB1, 10, 10, 30, 12, ES_LEFT | WS_CHILD |
             WS_VISIBLE | WS_BORDER
  AUTOCHECKBOX "Option 1", ID_CB1, 10, 40, 48, 12
  AUTOCHECKBOX "Option 2", ID_CB2, 10, 60, 48, 12
}
```

```
status.h:קובץ הכותר דרוש גם הוא לתוכנית
#define IDM_DIALOG  100
#define IDM_HELP    101
#define ID_UPDOWN   102
#define ID_EB1      103
#define ID_CB1      104
#define ID_CB2      105
#define ID_RESET    106
```



תרשים 14.1: דוגמת פלט של תוכנית שורת המצב

הפונקציה ()InitStatus שמופיעה בתוכנית, יוצרת את חלון המצב ומאתחלת אותו. שורת המצב מחולקת לשלושה חלקים שווים. חלוקת השורה מתבצעת בעזרת פונקציית API שנקראת ()GetClientRect. הפונקציה מקבלת את הגודל הנוכחי של אזור הלקוח של החלון שנבחר. זוהי תבנית הפקודה:

```
BOOL GetClientRect(HWND hwnd, LPRECT lpRect);
```

hwnd הוא הידית של החלון הנדון, ו-lpRec הוא מצביע אל המבנה RECT, אשר מקבל את מימדי אזור הלקוח של החלון.

שורת המצב מורכבת משלושה חלקים, ולכן רוחב תיבת הדו-שיח מחולק לשלושה (הרוחב מתקבל באמצעות הפונקציה ()GetClientRect). תוצאת החלוקה מתורגמת לנקודות הקצה של חלקי החלון, אשר מוצבות במערך parts. זכור, יש להעביר לחלון המצב את נקודת הקצה של כל אחד מהחלקים, **ולא** את רוחבם. לאחר שנקבעו חלקי השורה, ניתן להציג בהם את הטקסט.

הפונקציה ()DialogFunc מעדכנת את הטקסט שבכל אחד מהחלקים, בכל פעם שחל שינוי בפקד. הדבר נעשה על ידי משלוח הודעת SB_SETTEXT לחלק הקשור אל הפקד ששינה את מצבו.

נקודה נוספת אודות חלונות מצב: אם משנים את גודל חלון אב, הוא יקבל הודעת WM_SIZE. עליך לשלוח הודעת WM_SIZE אל חלון המצב בעזרת הפונקציה SendMessage(), כדי לאפשר את שינוי גודל חלון הבן במקביל. לדוגמה, אם ברצונך לאפשר שינוי אוטומטי בגודל חלון המצב שבדוגמה הקודמת, כאשר חל שינוי בגודל תיבת הדו-שיח, הוסף את משפט case הבא לפונקציה DialogFunc(). עליך, כמובן, ליצור את תיבת הדו-שיח בעזרת הסגנון WS_SIZEBOX.

```
case WM_SIZE:
  SendMessage(hStatusWnd, WM_SIZE, wPARAM, lParam);
  break;
```

תוכל לנסות להוסיף זאת בעצמך, אך זכור שהשיטה לא תגרום לחלקים עצמם לשנות את גודלם. השורה תתארך עד שתמלא את החלון. כדי לשנות את גודל החלקים, עליך לשלוח הודעת SB_SETPARTS בציון גודלי החלקים החדשים.

14.2 פקדי כרטיסיה

אחד הפקדים המעניינים ביותר מבחינה חזותית, הוא ***פקד הכרטיסיה*** (Tab Control). פקד הכרטיסיה מתפקד ככרטיסיה של תיקיית קובץ. בחירה בפקד זה מציגה את התיקיה הקשורה אליו בחזית התצוגה. הפעלתו של פקד הכרטיסיה קלה ופשוטה, אך מסובך במקצת לתכנת אותו. בסעיף זה נציג את את העקרונות הבסיסיים של פקד הכרטיסיה, ובסעיף הבא נמשיך ונעסוק בתכונות נוספות שלו.

יצירת פקד כרטיסיה

יוצרים פקד כרטיסיה באמצעות הפונקציה CreateWindow() או CreateWindowEx(), ובציון מחלקת החלון WC_TABCONTROL. פקד כרטיסיה יהיה בדרך כלל חלון-בן. הוא יווצר בעזרת הסגנון WS_VISIBLE, ולכן יוצג אוטומטית. הקוד הבא יוצר פקד כרטיסיה:

```
hTabWnd = CreateWindow(
                WC_TABCONTROL,
                "",
                WS_VISIBLE | WS_TABSTOP | WS_CHILD,
                0, 0, 100, 100,
                hwnd, /* handle of parent */
                NULL,
                hInst, /* instance handle */
                NULL
            );
```

לאחר יצירת הפקד, ניתן לשלוח הודעות מהיישום אל פקד הכרטיסיה, או בכיוון ההפוך, בכל פעם שניגשים אל הפקד.

פריטי כרטיסיה מוגדרים על ידי מבנה TC_ITEM, המוצג להלן:

```
typedef struct _TC_ITEM
{
  UINT mask;
  UINT lpReserved1;
  UINT lpReserved2;
  LPSTR pszText;
  int cchTextMax;
  int iImage;
  LPARAM lParam;
} TC_ITEM;
```

הערך שמכיל המשתנה mask קובע אם איברי המבנה pszText, iImage או lParam מכילים נתונים חוקיים, כאשר המבנה מקבל נתונים מפקד הכרטיסיה. המשתנה mask יכול להכיל אחד, או יותר מהערכים הבאים:

ערך במשתנה **mask**	משמעות
TCIF_ALL	pszText, iImage ו-lParam מכילים נתונים.
TCIF_IMAGE	iImage מכיל נתונים.
TCIF_PARAM	lParam מכיל נתונים.
TCIF_TEXT	pszText מכיל נתונים.

בעת הגדרת כרטיסיה, מצביע pszText אל המחרוזת שתוצג בה. כשמתקבל כל המידע אודות הכרטיסיה, יצביע pszText אל מערך שיקבל את הטקסט. במקרה זה, הערך שב-cchTextMax מציין את גודל המערך שאליו מצביע pszText.

אם הפקד קשור עם רשימת דמויות, אזי יכיל iImage את האינדקס של הדמות הקשורה לפקד שנבחר. אם הפקד אינו קשור לרשימת דמויות כלשהי, iImage חייב לקבל את הערך 1-.

lParam מכיל נתונים שמוגדרים על ידי היישום.

שיגור הודעות אל פקד כרטיסיה

הפונקציה ()SendMessage מאפשרת לשלוח מספר סוגי הודעות אל פקד הכרטיסיה. טבלה 14.2 מציגה חלק מההודעות הנפוצות ביותר. ההודעות נשלחות באופן תדיר, ולכן הונהגו פקודות מאקרו מיוחדות שמפשטות את שליחתן. להלן פקודות המאקרו המתאימות להודעות שבטבלה 14.2. בכל המקרים, hTabWnd היא הידית של פקד הכרטיסיה.

```
VOID TabCtrl_AdjustRect(HWND hTabWnd, BOOL operation, LPRECT
                    lpRect);
BOOL TabCtrl_DeleteAllItems(HWND hTabWnd);
BOOL TabCtrl_DeleteItem(HWND hTabWnd, int index);
int TabCtrl_GetCurSel(HWND hTabWnd);
BOOL TabCtrl_GetItem(HWND hTabWnd, int index, LPTC_ITEM item);
int TabCtrl_GetItemCount(HWND hTabWnd);
int TabCtrl_InsertItem(HWND , int index, CONST LPTC_ITEM item);
int TabCtrl_SetCurSel(HWND hTabWnd, int index);
BOOL TabCtrl_SetItem(HWND hTabWnd, int index, LPTC_ITEM item);
```

בעקרון, קל יותר להפעיל את המאקרוס מאשר לקרוא לפונקציה ()SendMessage.

טבלה 14.2: ההודעות הנפוצות ביותר של פקד הכרטיסיה

הודעה	משמעות
TCM_ADJUSTRECT	תרגום מימדי תצוגת הפקד לגודל החלון. wParam מציין את הפעולה. כשהוא מקבל ערך שונה מ-0, מתקבל מלבן החלון. כשהוא 0, מתקבל אזור התצוגה. lParam מצביע אל מבנה RECT שמכיל את הקואורדינטות של האזור המיועד לתרגום. בסיום הפעולה יכיל המבנה את הקואורדינטות המתורגמות.
TCM_DELETEALLITEMS	ביטול כל כרטיסיות הפקד. מחזירה ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה כישלון. wParam הוא 0. lParam הוא 0.
TCM_DELETEITEM	מוחקת את הכרטיסיה שנבחרה. מחזירה ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה כישלון. wParam מציין את האינדקס של הכרטיסיה המיועדת למחיקה. lParam הוא 0.
TCM_GETCURSEL	מחזירה את אינדקס הכרטיסיה הנוכחית שנבחרה. אם לא נבחרה כרטיסיה, תחזיר את הערך 1-. wParam הוא 0. lParam הוא 0.
TCM_GETITEMCOUNT	מחזירה את מספר הכרטיסיות. wParam הוא 0. lParam הוא 0.

הודעה	משמעות
TCM_GETITEM	מקבלת מידע על הכרטיסיה שנבחרה. מחזירה ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה כישלון. wParam מציין את האינדקס של הכרטיסיה. lParam מצביע אל מבנה TC_ITEM שמקבל את נתוני הכרטיסיה.
TCM_INSERTITEM	יוצרת (מכניסה) כרטיסיה חדשה. מחזירה ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה כישלון. wParam מציין את האינדקס של הכרטיסיה. lParam מצביע אל מבנה TC_ITEM שמתאר את הכרטיסיה.
TCM_SETCURSEL	בוחרת כרטיסיה. מחזירה את האינדקס של הכרטיסיה הקודמת שנבחרה. מחזירה 1- אם לא נבחרה כרטיסיה כלשהי לפני הכרטיסיה הנוכחית. wParam מציין את האינדקס של הכרטיסיה הנוכחית שנבחרה. lParam הוא 0.
TCM_SETITEM	קובעת נתונים בכרטיסיה שנבחרה. מחזירה ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה כישלון. wParam מציין את האינדקס של הכרטיסיה. lParam מצביע אל מבנה TC_ITEM שמכיל את המידע אודות הכרטיסיה.

פקד כרטיסיה אינו מכיל כרטיסיות כלשהן בעת יצירתו, ולכן התוכנית חייבת לשלוח אליו לפחות הודעת TCM_INSERTITEM אחת. קיימת הודעת פקד שלמרות שהיא שימושית, היא אינה כלולה באף אחת מהדוגמאות שבפרק זה: TCM_ADJUSTREC. הודעה זו מיועדת לקבל את מימדי אזור התצוגה של פקד הכרטיסיה. זכור, החלון של פקד הכרטיסיה שאתה יוצר כולל את הכרטיסיות וגם את האזור שבו יוצג המידע, או תיפתח תיבת דו-שיח. אזור התצוגה הוא החלק של חלון הפקד שאינו כולל את הכרטיסיה (כלומר, האזור בו ניתן להציג פריטים אחרים). מאחר שזה האזור בו מוצג מידע הקשור לכרטיסיה, תצטרך לדעת את מימדיו.

הודעות Notification של הכרטיסיה

כל גישה אל פקד הכרטיסיה יוצרת הודעת WM_NOTIFY (או Tab Notification Message). פקדי כרטיסיה יכולים ליצור שני סוגי קודים של **הודעת שינוי בחירה** (Selction-Change): TCN_SELCHANGE ו-TCN_SELCHANGING. ההודעה TCN_SELCHANGING נשלחת כאשר בחירת הכרטיסיה עומדת להשתנות; TCN_SELCHANGE נשלחת לאחר שנבחרה כרטיסיה חדשה.

עם קבלת הודעת WM_NOTIFY, יצביע lParam אל המבנה NMHDR. השדה code המהווה חלק ממבנה זה יכיל את קוד הnotification. הידית של פקד הכרטיסיה שיוצר את ההודעה, נמצאת בשדה hwndFrom.

תוכנית הדגמה פשוטה של פקד כרטיסיה

לפניך תוכנית **TabControl** קצרה ופשוטה להדגמה של פקד כרטיסיה. (התוכנית נמצאת בתקליטור chap14\TabControl) . התוכנית יוצרת פקד ובו שלוש כרטיסיות. הכרטיסיות מסומנות בטקסט One, Two ו-Three. כשבוחרים כרטיסיה חדשה, משתקפת העובדה בהצגת הודעה מתאימה. תרשים 14.2 מציג דוגמת פלט של התוכנית.

```
#include <windows.h>
#include <commctrl.h>
#include <stdio.h>

#include "TabControl.h"
#define IS_WIN32 TRUE

HINSTANCE hInst;         // current instance
HWND hWnd;
HWND hTabWnd;
LPCTSTR lpszAppName = "Generic";
LPCTSTR lpszTitle =   "Generic Application";
BOOL RegisterWin95(CONST WNDCLASS* lpwc);

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE
                   hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
 {
   TC_ITEM tci;
   MSG msg;
   WNDCLASS wc;
   RECT WinDim;

   wc.style           = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc     = (WNDPROC)WndProc;
   wc.cbClsExtra      = 0;
   wc.cbWndExtra      = 0;
   wc.hInstance       = 0;
   wc.hIcon           = LoadIcon(hInstance, lpszAppName);
   wc.hCursor         = LoadCursor(NULL, IDC_ARROW);
```

```
  wc.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
  wc.lpszMenuName   = lpszAppName;
  wc.lpszClassName  = lpszAppName;

  if(!RegisterWin95(&wc))
     return false;
  hInst = hInstance; /* save the current instance handle */

  hWnd = CreateWindow (lpszAppName,
                       lpszTitle,
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, 0,
                       CW_USEDEFAULT, 0,
                       NULL,
                       NULL,
                       hInstance,
                       NULL
                      );

GetClientRect(hWnd, &WinDim);  /* get size of parent window */
InitCommonControls();

/* create a tab control */
hTabWnd = CreateWindow(
                WC_TABCONTROL,
                "",
                WS_VISIBLE | WS_TABSTOP | WS_CHILD,
                0, 0, WinDim.right, WinDim.bottom,
                hWnd,
                NULL,
                hInst,
                NULL
           );

tci.mask = TCIF_TEXT;

tci.iImage = -1;

tci.pszText = "One";
TabCtrl_InsertItem(hTabWnd, 0, &tci);
```

```
  tci.pszText = "Two";
  TabCtrl_InsertItem(hTabWnd, 1, &tci);

  tci.pszText = "Three";
  TabCtrl_InsertItem(hTabWnd, 2, &tci);

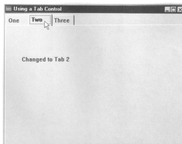
  /* Display the window. */
  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  /* Create the message loop. */
  while(GetMessage(&msg, NULL, 0, 0))
  {
    TranslateMessage(&msg); /* allow use of keyboard */
    DispatchMessage(&msg); /* return control to Windows */
  }
  return msg.wParam;
}

BOOL RegisterWin95(CONST WNDCLASS* lpwc)
 {
   WNDCLASSEX wcex;

   wcex.style           = lpwc->style;
   wcex.lpfnWndProc     = lpwc->lpfnWndProc;
   wcex.cbClsExtra      = lpwc->cbClsExtra;
   wcex.cbWndExtra      = lpwc->cbWndExtra;
   wcex.hInstance       = lpwc->hInstance;
   wcex.hIcon           = lpwc->hIcon;
   wcex.hCursor         = lpwc->hCursor;
   wcex.hbrBackground   = lpwc->hbrBackground;
   wcex.lpszMenuName    = lpwc->lpszMenuName;
   wcex.lpszClassName   = lpwc->lpszClassName;
   wcex.cbSize          = sizeof(WNDCLASSEX);
   wcex.hIconSm         = LoadIcon(wcex.hInstance, "SMALL");
   return RegisterClassEx(&wcex);
 }
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
                         WPARAM wParam, LPARAM lParam)
{
  NMHDR *nmptr;
  int tabnumber;
  HDC hdc;
  char str[80];
  switch(message) {
      case WM_COMMAND:
         switch(LOWORD(wParam))
          {
            case IDM_TEST :
               break;
            case IDM_EXIT :
               DestroyWindow(hWnd);
               break;
          }
         break;
      case WM_NOTIFY: /* process a tab change */
      nmptr = (LPNMHDR) lParam;
      if(nmptr->code == TCN_SELCHANGE) {
        tabnumber = TabCtrl_GetCurSel((HWND)nmptr->hwndFrom);
        hdc = GetDC(hTabWnd);
        sprintf(str, "Changed to Tab %d", tabnumber+1);
        SetBkColor(hdc, RGB(200, 200, 200));
        TextOut(hdc, 40, 100, str, strlen(str));
        ReleaseDC(hTabWnd, hdc);
      }
      break;
    case WM_DESTROY: /* terminate the program */
      PostQuitMessage(0);
      break;
    default:
     /* Let Windows 95 process any messages not specified in
        the preceding switch statement. */
    return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
```



תרשים 14.2: פלט לדוגמה של התוכנית הראשונה של פקד הכרטיסיה

טרם יצירת הפקד, קוראת התוכנית לפונקציה ()GetClientRect, כדי לקבל את גודל החלון הראשי. הפקד שנוצר לאחר מכן מתאים בגודלו לאזור הלקוח של חלון האב וממלא אותו בשלמותו. הדבר נעשה באופן שרירותי, אך מקובל למדי. לאחר יצירת פקד הכרטיסיות, נוצרות בו שלוש כרטיסיות.

הפונקציה ()WndProc מציגה הודעה באזור התצוגה של הכרטיסיה, ומדווחת על בחירת כרטיסיה חדשה, כל פעם שההודעה WM_NOTIFY מקבלת קוד מסוג TCN_SELCHANGE.

14.3 הפעלת פקדי כרטיסיה

קל להפעיל פקדי כרטיסיה, אך יצירתם מסובכת למדי, מכיון שלכל כרטיסיה קשורה בדרך כלל תיבת דו-שיח. כל פעם שבוחרים בכרטיסיה חדשה, יש להסיר מהתצוגה את תיבת הדו-שיח הנוכחית, ולהציג תחתיה תיבת דו-שיח חדשה. כמו כן, לרוב צריך להתאים את תיבות הדו-שיח השונות לאזור התצוגה של פקד הכרטיסיה. פקדי כרטיסיה ותיבות הדו-שיח שלהם כרוכים בנושאים נוספים ומסובכים שאינם כלולים בספר. למרות זאת, נלמד בסעיף זה שיטה כללית לשילוב פקד הכרטיסיה ביישומים. אם תבין את העקרונות הבסיסיים, תוכל להוסיף ולשפר בעצמך את פקדי הכרטיסיה ביישומים שתכתוב.

העובדה שניתן לבחור כרטיסיה חדשה בכל עת, מרמזת על כך שיש לשלב תיבות דו-שיח **לא-מודאליות** (Modeless). תיבת דו-שיח לא-מודאלית מאפשרת להפעיל חלקים אחרים ביישום מבלי לסגור אותה לפני כן (בניגוד לתיבת דו-שיח מודאלית, אותה יש לסגור כדי להפעיל חלקים אחרים בתוכנית). בדרך כלל, כשבוחרים כרטיסיה חדשה, היישום סוגר את תיבת הדו-שיח המוצגת ולאחר מכן מפעיל את התיבה הבאה. ניתן לסגור תיבת דו-שיח בכל עת, ולכן צריך לנקוט בפעולה המתאימה (כגון שמירת ההגדרות שלה) בטרם מציגים את התיבה הבאה.

נעבור לתוכנית **TabCtl2** (בתקליטור chap14\TabCtl2) אשר תאפשר לנו ללמוד כיצד להפעיל פקד כרטיסיה. התוכנית יוצרת פקד ובו שלוש כרטיסיות: Options, Pitch ו-Page Size. כל אחת מהכרטיסיות קשורה לתיבת דו-שיח לא-מודאלית. הראשונה היא מהסוג שיצרנו בתחילת הפרק עבור **שורת המצב** (Status Bar). שתי הכרטיסיות הנוספות הן ריקות ולמעשה, רק **תופסות מקום** (Placeholders), אותן שילבנו לצורך ההדגמה. תרשים 14.3 מציג פלט לדוגמה של התוכנית.

```
#include <windows.h>
#include <commctrl.h>
#include <stdio.h>
#include "TabCtl2.h"

#define IS_WIN32 TRUE
#define NUMPARTS 3

BOOL CALLBACK DlgFunc1(HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK DlgFunc2(HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK DlgFunc3(HWND, UINT, WPARAM, LPARAM);
void InitStatus(HWND hWnd);

HINSTANCE hInst;         // current instance
HWND hWnd;
HWND hTabWnd;
HWND hStatusWnd;
HWND hDlg = (HWND) NULL;
LPCTSTR lpszAppName = "Generic";
LPCTSTR lpszTitle =  "Generic Application";
BOOL RegisterWin95(CONST WNDCLASS* lpwc);

int parts[NUMPARTS];

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE
                  hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
 {
   TC_ITEM tci;
   MSG msg;
   WNDCLASS wc;

   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = (WNDPROC)WndProc;
   wc.cbClsExtra     = 0;
```

```
wc.cbWndExtra     = 0;
wc.hInstance      = 0;
wc.hIcon          = LoadIcon(hInstance, lpszAppName);
wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
wc.lpszMenuName   = lpszAppName;
wc.lpszClassName  = lpszAppName;

if(!RegisterWin95(&wc))
   return false;
hInst = hInstance; /* save the current instance handle */

hWnd = CreateWindow (lpszAppName,
                     lpszTitle,
                     WS_OVERLAPPEDWINDOW,
                     CW_USEDEFAULT, 0,
                     CW_USEDEFAULT, 0,
                     NULL,
                     NULL,
                     hInstance,
                     NULL
                    );
if(!hWnd)
   return false;

InitCommonControls();

hTabWnd = CreateWindow(
           WC_TABCONTROL,
           "",
           WS_VISIBLE | WS_TABSTOP | WS_CHILD,
           20, 20, 325, 250,
           hWnd,
           NULL,
           hInst,
           NULL
          );
```

```
   tci.mask = TCIF_TEXT;

   tci.iImage = -1;

   tci.pszText = "Options";
   TabCtrl_InsertItem(hTabWnd, 0, &tci);

   tci.pszText = "Pitch";
   TabCtrl_InsertItem(hTabWnd, 1, &tci);

   tci.pszText = "Page Size";
   TabCtrl_InsertItem(hTabWnd, 2, &tci);

   hDlg = CreateDialog(hInst, "MYDB1", hTabWnd, DlgFunc1);

   /* Display the window. */
   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
   /* Create the message loop. */
   while(GetMessage(&msg, NULL, 0, 0))
  {
    TranslateMessage(&msg); /* allow use of keyboard */
    DispatchMessage(&msg); /* return control to Windows */
  }
  return msg.wParam;
}

/* This function is called by Windows 95 and is passed
   messages from the message queue.*/
LRESULT CALLBACK WndProc(HWND hwnd, UINT message,
                         WPARAM wParam, LPARAM lParam)
{
  NMHDR *nmptr;
  int tabnumber = 0;
  switch(message) {
    case WM_COMMAND:
         switch(LOWORD(wParam))
          {
            case IDM_TEST :
               break;
```

```
            case IDM_EXIT :
               DestroyWindow(hWnd);
               break;
          }
         break;
    case WM_NOTIFY:
      nmptr = (LPNMHDR) lParam;
      if(nmptr->code == TCN_SELCHANGE) {
        if(hDlg) DestroyWindow(hDlg);
        tabnumber = TabCtrl_GetCurSel((HWND)nmptr->hwndFrom);
        switch(tabnumber) {
          case 0:
            hDlg = CreateDialog(hInst, "MYDB1",
                                hTabWnd, DlgFunc1);
            break;
          case 1:
            hDlg = CreateDialog(hInst, "MYDB2",
                                hTabWnd, DlgFunc2);
            break;
          case 2:
            hDlg = CreateDialog(hInst, "MYDB3",
                                hTabWnd, DlgFunc3);
            break;
        }
      }
      break;
    case WM_DESTROY: /* terminate the program */
      if(hDlg) DestroyWindow(hDlg);
      PostQuitMessage(0);
      break;
    default:
     /* Let Windows 95 process any messages not specified in
        the preceding switch statement. */
     return DefWindowProc(hwnd, message, wParam, lParam);
  }
  return 0;
}
```

```
/* First dialog function. */
BOOL CALLBACK DlgFunc1(HWND hDlg, UINT message,
                        WPARAM wParam, LPARAM lParam)
{
  static long udpos = 0;
  static char str[80];
  static HWND hEboxWnd;
  static HWND udWnd;
  static statusCB1, statusCB2;
  int low=0, high=20;

  switch(message) {
    case WM_INITDIALOG:
      InitStatus(hDlg);

      hEboxWnd = GetDlgItem(hDlg, ID_EB1);
      udWnd = CreateUpDownControl(
                    WS_CHILD | WS_BORDER | WS_VISIBLE |
                    UDS_SETBUDDYINT | UDS_ALIGNRIGHT,
                    10, 10, 50, 50,
                    hDlg,
                    ID_UPDOWN,
                    hInst,
                    hEboxWnd,
                    high, low, high/2);
     return 1;
    case WM_VSCROLL: /* process up/down control */
      if(udWnd==(HWND)lParam) {
        udpos = GetDlgItemInt(hDlg, ID_EB1, NULL, 1);
        sprintf(str, "Up-down: %d", udpos);
        SendMessage(hStatusWnd, SB_SETTEXT,
                    (WPARAM) 0, (LPARAM) str);
      }
      return 1;
    case WM_COMMAND:
      switch(LOWORD(wParam)) {
```

```
        case ID_CB1: /* process check box 1 */
          statusCB1 = SendDlgItemMessage(hDlg, ID_CB1,
                                        BM_GETCHECK, 0, 0);
          if(statusCB1) sprintf(str, "Option 1: ON");
          else sprintf(str, "Option 1: OFF");
          SendMessage(hStatusWnd, SB_SETTEXT,
                      (WPARAM) 1, (LPARAM) str);

          return 1;
        case ID_CB2: /* process check box 2 */
          statusCB2 = SendDlgItemMessage(hDlg, ID_CB2,
                                        BM_GETCHECK, 0, 0);
          if(statusCB2) sprintf(str, "Option 2: ON");
          else sprintf(str, "Option 2: OFF");
          SendMessage(hStatusWnd, SB_SETTEXT,
                      (WPARAM) 2, (LPARAM) str);
          return 1;
        case ID_RESET: /* reset options */
          SendMessage(udWnd, UDM_SETPOS, 0, (LPARAM) high /2);
          SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 0,
                      (LPARAM) "Up-down: 10");
          SendDlgItemMessage(hDlg, ID_CB1,
                             BM_SETCHECK, 0, 0);
          SendDlgItemMessage(hDlg, ID_CB2,
                             BM_SETCHECK, 0, 0);
          SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 1,
                      (LPARAM) "Option 1: OFF");
          SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 2,
                      (LPARAM) "Option 2: OFF");
          return 1;
        case IDCANCEL:
        case IDOK:
          PostQuitMessage(0);
          return 1;
      }
  }
  return 0;
}
```

```
/* Second dialog function. This is just a placeholder. */
BOOL CALLBACK DlgFunc2(HWND hdwnd, UINT message,
                       WPARAM wParam, LPARAM lParam)
{
  switch(message) {
    case WM_COMMAND:
      switch(LOWORD(wParam)) {
        case IDOK:
          PostQuitMessage(0);
          return 1;
      }
  }
  return 0;
}

/* Third dialog function. This is just a placeholder. */
BOOL CALLBACK DlgFunc3(HWND hdwnd, UINT message,
                       WPARAM wParam, LPARAM lParam)
{
  switch(message) {
    case WM_COMMAND:
      switch(LOWORD(wParam)) {
        case IDOK:
          PostQuitMessage(0);
          return 1;
      }
  }
  return 0;
}

/* Initialize the status bar. */
void InitStatus(HWND hWnd)
{
  RECT WinDim;
  int i;
  GetClientRect(hWnd, &WinDim);

  for(i=1; i<=NUMPARTS; i++)
    parts[i-1] = WinDim.right/NUMPARTS * i;
```

```
  /* Create a status bar */
  hStatusWnd = CreateWindow(STATUSCLASSNAME,
                "", /* not used in this example */
                WS_CHILD | WS_VISIBLE,
                0, 0, 0, 0,
                hWnd,
                NULL,
                hInst,
                NULL );

  SendMessage(hStatusWnd, SB_SETPARTS,
              (WPARAM) NUMPARTS, (LPARAM) parts);
  SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 0,
              (LPARAM) "Up-down: 10");
  SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 1,
              (LPARAM) "Option 1: OFF");
  SendMessage(hStatusWnd, SB_SETTEXT, (WPARAM) 2,
              (LPARAM) "Option 2: OFF");
}

BOOL RegisterWin95(CONST WNDCLASS* lpwc)
 {
   WNDCLASSEX wcex;

   wcex.style           = lpwc->style;
   wcex.lpfnWndProc     = lpwc->lpfnWndProc;
   wcex.cbClsExtra      = lpwc->cbClsExtra;
   wcex.cbWndExtra      = lpwc->cbWndExtra;
   wcex.hInstance       = lpwc->hInstance;
   wcex.hIcon           = lpwc->hIcon;
   wcex.hCursor         = lpwc->hCursor;
   wcex.hbrBackground   = lpwc->hbrBackground;
   wcex.lpszMenuName    = lpwc->lpszMenuName;
   wcex.lpszClassName   = lpwc->lpszClassName;
   wcex.cbSize          = sizeof(WNDCLASSEX);
   wcex.hIconSm         = LoadIcon(wcex.hInstance, "SMALL");
   return RegisterClassEx(&wcex);
 }
```

התוכנית זקוקה לקובץ המשאבים הבא:

```
#include <windows.h>
#include "tab.h"

MYDB1 DIALOG 2, 16, 158, 106
STWS_CHILD | WS_VISIBLE | WS_BORDER
{
  PUSHBUTTON "Reset", ID_RESET, 92, 34, 37, 14,
             WS_CHILD | WS_VISIBLE | WS_TABSTOP
  PUSHBUTTON "OK", IDOK, 92, 53, 37, 14,
             WS_CHILD | WS_VISIBLE | WS_TABSTOP
  EDITTEXT ID_EB1, 10, 10, 30, 12, ES_LEFT | WS_CHILD |
             WS_VISIBLE | WS_BORDER
  AUTOCHECKBOX "Option 1", ID_CB1, 10, 40, 48, 12
  AUTOCHECKBOX "Option 2", ID_CB2, 10, 60, 48, 12
}

MYDB2 DIALOG 2, 16, 158, 106
STYLE WS_CHILD | WS_VISIBLE | WS_BORDER
{
  PUSHBUTTON "OK", IDOK, 92, 53, 37, 14,
             WS_CHILD | WS_VISIBLE | WS_TABSTOP
  LTEXT "Choose Pitch", 1, 10, 10, 60, 12
  AUTORADIOBUTTON "High Pitch", ID_RB1, 10, 40, 48, 12
  AUTORADIOBUTTON "Medium Pitch", ID_RB2, 10, 60, 52, 12
  AUTORADIOBUTTON "Low Pitch", ID_RB2, 10, 80, 48, 12
}

MYDB3 DIALOG  2, 16, 158, 106
STYLE WS_CHILD | WS_VISIBLE | WS_BORDER
{
  PUSHBUTTON "OK", IDOK, 92, 53, 37, 14,
             WS_CHILD | WS_VISIBLE | WS_TABSTOP
  LTEXT "Choose  Page Size", 2, 10, 10, 70, 12
  AUTORADIOBUTTON "Small", ID_RB3, 10, 40, 48, 12
  AUTORADIOBUTTON "Medium", ID_RB4, 10, 60, 48, 12
  AUTORADIOBUTTON "Large", ID_RB5, 10, 80, 48, 12
}
```

לפניך קובץ הכותר tab.h :

```
#define IDM_DIALOG  100
#define IDM_HELP    101
#define ID_UPDOWN   102
#define ID_EB1      103
#define ID_CB1      104
#define ID_CB2      105
#define ID_RESET    106
#define ID_RB1      107
#define ID_RB2      108
#define ID_RB3      109
#define ID_RB4      110
#define ID_RB5      111
```

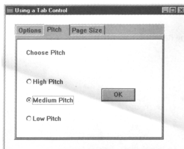
החלק המעניין בתוכנית נמצא במשפט case WM_NOTIFY, של הפונקציה ()WndProc, שמוצג כאן פעם נוספת לנוחותך :

```
case WM_NOTIFY:
  nmptr = (LPNMHDR) lParam;
  if(nmptr->code == TCN_SELCHANGE) {
    if(hDlg) DestroyWindow(hDlg);
    tabnumber = TabCtrl_GetCurSel((HWND)nmptr->hwndFrom);
    switch(tabnumber) {
      case 0:
        hDlg = CreateDialog(hInst, "MYDB1",
                     hTabWnd, DialogFunc1);
        break;
      case 1:
        hDlg = CreateDialog(hInst, "MYDB2",
                     hTabWnd, DialogFunc2);
        break;
      case 2:
        hDlg = CreateDialog(hInst, "MYDB3",
                     hTabWnd, DialogFunc3);
        break;
    }
  }
  break;
```

שני אירועים קורים בכל פעם שבוחרים כרטיסיה חדשה. הפונקציה ()DestroyWindow מסירה מהמסך את תיבת הדו-שיח הקשורה לכרטיסיה המוצגת (כידוע, לסגירת תיבת דו-שיח לא-מודאלית יש לקרוא לפונקציה ()DestroyWindow, ולא ל-()EndDialog,

כמקובל בתיבות דו-שיח מודאליות). בשלב הבא מתקבלת הבחירה בכרטיסיה הנוכחית והפונקציה ()CreateDialog יוצרת את תיבת הדו-שיח שלה (זכור, זו הדרך ליצור תיבות דו-שיח לא-מודאליות).

בטרם תתקדם, עליך להפעיל את התוכנית במצבים שונים, כשאתה משנה את תיבות הדו-שיח ופקדי הכרטיסיה. ניתן לקשר **תוויות לחצן** (Tooltips) לפקדי כרטיסיה, משימה רבת-אתגר שתוכל לנסות בעצמך.



תרשים 14.3: דוגמת פלט של התוכנית השנייה של פקד הכרטיסיה

14.4 פקדי תצוגת עץ

הפקד האחרון שיסוקר בפרק זה הוא **פקד תצוגת עץ** (Tree View Control). פקד זה משמש להצגת מידע באמצעות מבנה עץ. רשימת הקבצים של Windows Explorer, מהווה דוגמה לפקד תצוגת עץ. עץ מרמז על מבנה היררכי של נתונים, ולכן יש להשתמש בו להצגת מידע היררכי בלבד. פקדי תצוגת עץ הינם רבי-עוצמה ותומכים במגוון אפשרויות גדול. למעשה, ניתן לכתוב ספר שלם שיעסוק בפקדי תצוגת עץ בלבד! לכן, יתמקד סעיף זה ביסודות של פקדי תצוגה אלה. כאשר העקרונות יהיו נהירים לך, תוכל לשכלל את השימוש בפקדי תצוגת עץ ולשלב בהם רכיבים נוספים.

יצירת פקד תצוגת עץ

פקד תצוגת עץ הוא חלון אשר נוצר באמצעות הפונקציה ()CreateWindow או ()CreateWindowEx, ומבוסס על המחלקה WC_TREEVIEW. פקד תצוגת עץ הוא בדרך כלל חלון בן שנוצר באמצעות הסגנון WS_VISIBLE, ולכן הוא מוצג באופן אוטומטי. WS_TABSTOP אף הוא כלול אוטומטית בפקד. תצוגות עץ מאפשרות לכלול סגנונות קרובים בעת יצירתם, ראה למשל בדוגמה הבאה.

סגנון	משמעות
TVS_HASLINES	שורות מקשרות בין ענפי העץ.
TVS_LINESATROOT	שורות מקשרות בין השורש והענפים.
TVS_HASBUTTONS	לחצני כיווץ/הרחבה משמאל לכל ענף.

שילוב הסגנונות TVS_HASLINES ו- TVS_LINESATROOT גורם להצגת שורות לכל אחד מהמרכיבים בעץ, וכך למעשה מתקבל המראה האופייני של העץ. שילוב הסגנון TVS_HASBUTTONS מאפשר להוסיף את לחצני ההרחבה והכיווץ התקניים. לחצנים אלה מציגים את הסימן + אם ניתן להרחיב את הענף להצגת רמה נוספת אחת לפחות, או את הסימן - במקרה שהענף הורחב במלואו. ניתן גם ללחוץ על הלחצנים כדי להרחיב את הענף או לכווץ אותו. בעת יצירת הפקד נכללים כל שלושת הסגנונות. הקוד הבא יוצר חלון תצוגת עץ תקנית:

```
hTreeWndCtrl = CreateWindow(
                WC_TREEVIEW,
                "",
                WS_VISIBLE | WS_TABSTOP | WS_CHILD |
                TVS_HASLINES | TVS_HASBUTTONS |
                TVS_LINESATROOT,
                0, 0, 100, 100,
                hwnd, /* handle of parent */
                NULL,
                hInst, /* instance handle */
                NULL
           );
```

ברגע יצירתו, פקד תצוגת העץ ריק מפריטים. בסעיף הבא נתאר כיצד למלא את העץ בתוכן.

שיגור הודעות אל תצוגת עץ

פקדי תצוגת עץ מגיבים להודעות אחדות. טבלה 14.3 מציגה כמה מהנפוצות שבהן. תדירות השימוש בהודעות פקד תצוגת העץ גבוהה מאוד, ולכן קיימות לשם כך פקודות מאקרו מיוחדות. להלן פקודות המאקרו שמקבילות להודעות בטבלה 14.3. בכל המקרים, hTreeWnd מכיל את הידית של פקד הכרטיסיה.

```
BOOL TreeView_DeleteItem(HWND hTreeWnd, HTREEITEM hItem)
BOOL TreeView_Expand(HWND hTreeWnd, HTREEITEM hItem, UINT action)
BOOL TreeView_GetItem(HWND hTreeWnd, LPTV_ITEM hItem)
HTREEVIEW TreeView_InsertItem(HWND hTreeWnd, LPTV_INSERTSTRUCT
                              item)
BOOL TreeView_Select(HWND hTreeWnd, HTREEITEM hItem, UINT action)
```

תוכנית הדוגמה שבפרק מכילה שתי הודעות בלבד, TVM_INSERTITEM ו-TVM_EXPAND. מובן שהיישום שלך יכול להכיל הודעות נוספות.

טבלה 14.3: הודעות נפוצות של תצוגת העץ

הודעה	משמעות
TVM_DELETEITEM	מוחקת פריט מתוך רשימת העץ. מחזירה ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה כישלון. lParam הוא 0. lParam מכיל את הידית של הפריט המיועד למחיקה.
TVM_EXPAND	מרחיבה או מכווצת את רשימת העץ ברמה אחת. מחזירה ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה כישלון. wParam מציין את הפעולה. הפעולה חייבת להיות אחת מהפעולות הבאות: TVECOLLAPSE (כיווץ העץ),TVECOLLAPSERESET (כיווץ העץ ומחיקת פריטי הבנים), TVE_EXPAND (הרחבת העץ), או TVE_TOGGLE (מעבר בין שני מצבים). lParam מכיל את הידית של הענף ברמה הגבוהה יותר (ענף אב).
TVM_GETITEM	מקבלת את מאפייני הפריט. מחזירה ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה כישלון. wParam הוא 0. lParam מכיל את המצביע אל מבנה TV_ITEM שמקבל את המידע אודות הפריט.
TVM_INSERTITEM	מכניסה פריט לעץ. מחזירה ידית אל הפריט שהוכנס לעץ, או NULL במקרה שהפעולה נכשלה. wParam הוא 0. lParam מכיל את המצביע אל מבנה TV_INSERTSTRUCT שמכיל את המידע אודות הפריט.
SELECTITEM_TVM	בוחרת פריט מתוך תצוגת העץ. מחזירה ערך שונה מ-0 במקרה של הצלחה, ו-0 במקרה כישלון. wParam מציין את הפעולה שמתייחסת לפריט שנבחר. במקרה של TVGN_CART, הפעולה תהיה בחירת פריט. במקרה של TVGN_DROPHILITE, הפריט יסומן לצורך גרירה ושחרור (Drag-And-Drop). במקרה של TVGN_FIRSTVISIBLE, התצוגה תיגלל כדי שהפריט שנבחר יופיע בראש הרשימה. lParam מכיל את ידית הפריט.

כשפריט מסוים נבחר, המידע עליו יופיע במבנה TV_INSERTSTRUCT הבא:

```
typedef struct _TV_INSERTSTRUCT {
  HTREEITEM hParent;
  HTREEITEM hInsertAfter;
  TV_ITEM item;
} TV_INSERTSTRUCT;
```

hParent היא ידית אל אב הפריט. אם לפריט אין אב, אזי שדה זה צריך להכיל את TVI_ROOT. הערך ב-hInsertItem קובע את אופן הכנסת פריטים לעץ. אם הוא מכיל את הידית של הפריט, הפריט החדש ייכנס לעץ לאחר הפריט. אחרת, hInsertItem יכול לקבל אחד מהערכים הבאים:

ערך המשתנה hInsertItem	משמעות
TVI_FIRST	הכנסת פריט חדש לראש הרשימה
TVI_LAST	הכנסת פריט חדש לסוף הרשימה
TVI_SORT	הכנסת פריט חדש לפי סדר הא״ב

התוכן של item מתאר את הפריט. זהו מבנה TV_ITEM, שמתואר להלן:

```
typedef struct _TV_ITEM {
  UINT mask;
  HTREEITEM hItem;
  UINT state;
  UINT stateMask;
  LPSTR pszText;
  int cchTextMax;
  int iImage;
  int iSelectedImage;
  int cChildren;
  LPARAM lParam;
} TV_ITEM;
```

הערך שמכיל המשתנה mask קובע מי מאיברי המבנה TV_ITEM מכיל נתונים חוקיים כשהמבנה מקבל מידע מפקד תצוגת העץ. להלן הערכים שהוא יכול לקבל:

ערך במשתנה mask	המשתנה (או משתנים) שמכיל(ים) את הנתונים
TVIF_HANDLE	hItem
TVIF_STATE	stateMask ,state
TVIF_TEXT	cchTextMax ,pszText

המשתנה (או משתנים) שמכיל(ים) את הנתונים	ערך במשתנה mask
iImage	TVIF_IMAGE
iSelectedImage	TVIF_SELECTEDIMAGE
cChildren	TVIF_CHILDREN
lParam	TVIF_LPARAM

האיבר state במבנה מכיל את המצב של פקד תצוגת העץ. להלן מספר מצבים נפוצים:

משמעות	סטטוס
פריט מבוטל	TVIS_DISABLED
פריט מסומן לצורך פעולת גרירה ושחרור	TVIS_DROPHILITED
הענף שמסתעף מהפריט הורחב במלואו (ישים לפריטי אב בלבד).	TVIS_EXPAND
הענף שמסתעף מהפריט הורחב כדי רמה אחת, או יותר (ישים לפריטי אב בלבד).	TVIS_EXPANDEDONCE
פריט בפוקוס.	TVIS_FOCUSED
פריט נבחר.	TVIS_SELECTED

המשתנה stateMask קובע איזה מצב כרטיסיה יש להציב, או לקבל. זה יכול להיות אחד, או יותר מהערכים בטבלה הקודמת.

כשמכניסים פריט לעץ, pszText מצביע אל המחרוזת שתוצג בעץ. כשמתקבל המידע הנוגע לפריט, חייב pszText להצביע אל המערך שיקבל את הטקסט. במקרה זה מציין cchTextMax את גודל המערך ש-pszText מצביע אליו. אחרת, אין התייחסות ל-cchTextMax.

אם קיימת רשימת דמויות שמקושרת עם פקד הכרטיסיה, iImage יכיל את האינדקס של התמונה הקשורה אל פקד הכרטיסיה. אם לא קיימת רשימת תמונות, iImage יכיל את הערך 1-. iSelectedImage מכילה את הסמל שנבחר מתוך הרשימה, אם קיים כזה.

כאשר מתקבל מידע אודות פריט כלשהו, יציין cChildren את מספר הבנים הקשורים עימו.

lParam מכיל נתונים שמוגדרים על ידי היישום.

הודעות של תצוגת העץ

כשנכנסים אל פקד תצוגת עץ נוצרות הודעות WM_NOTIFY, כלומר, העץ העץ שולח הודעות (Notification Messages). קיימות מספר הודעות מהעץ הקשורות אל פקדי תצוגת עץ. להלן ההודעות הנפוצות ביותר:

הודעת notification	משמעות
TVN_DELETEITEM	פריט נמחק.
TVN_ITEMEXPANDING	ענף עומד להתרחב, או להתכווץ.
TVN_ITEMEXPANDED	ענף הורחב, או כווץ.
TVN_SELCHANGING	פריט חדש עומד להיבחר.
TVN_SELCHANGED	נבחר פריט חדש.

כשמתקבלת הודעת WM_NOTIFY, יצביע lParam אל המבנה NM_TREEVIEW. לפניך הקוד של המבנה NM_TREEVIEW:

```
typedef struct _NM_TREEVIEW {
  NMHDR hdr;
  UINT action;
  TV_ITEM itemOld;
  TV_ITEM itemNew;
  POINT ptDrag;
} NM_TREEVIEW;
```

השדה הראשון של NM_TREEVIEW הוא המבנה התקני NMHDR. קוד ההודעה ייכנס אל השדה code של NMHDR. השדה hwndFrom באותו מבנה יכיל את הידית של פקד העץ אשר הפיק את ההודעה.

השדה action מכיל מידע ייחודי להודעה מהפקד. המבנים itemOld ו-itemNew מכילים מידע אודות הפריט הקודם שנבחר (אם ישים) ושל הפריט החדש שנבחר (אם ישים). ptDrag מכיל את נתוני המיקום של העכבר בעת הפקת ההודעה.

במקרה של ההודעות TVN_SELCHANGING ו-TVN_SELCHANGED, מתאר itemOld את הפריט שנבחר קודם, ו-itemNew מתאר את זה שנבחר זה עתה. במקרה של ההודעות TVN_ITEMEXPANDING ו-TVN_ITEMEXPANDED, האיבר itemNew מתאר את הפריט שהוא האב של הענף שהורחב. במקרה של ההודעה TVN_DELETEITEM מתאר itemOld את הפריט שנמחק.

תוכנית הדגמה של תצוגת עץ

התוכנית **Tree_Ctl** הבאה (בתקליטור Chap14\Tree_Ctl) מדגימה פקד תצוגת עץ. היא יוצרת פקד ולאחר מכן מכניסה אליו חמישה פריטים. התוכנית גם כוללת תפריט שמאפשר להרחיב ענף בודד, את העץ כולו, או לכווץ ענף יחיד. כל בחירה של ענף חדש בעץ באה לידי ביטוי בחלון התוכנית. תרשים 14.4 מציג דוגמת פלט של התוכנית.

```
#include <windows.h>
#include <commctrl.h>
#include <string.h>

#include "Tree_Ctl.h"
#if defined (win32)
   #define IS_WIN32 TRUE
#else
   #define IS_WIN32 FALSE
#endif

#define NUM 5

HINSTANCE hInst;         // current instance
LPCTSTR lpszAppName = "Generic";
LPCTSTR lpszTitle =   "Generic Application";
BOOL RegisterWin95(CONST WNDCLASS* lpwc);

void InitTree(void);
void report(HDC hdc, char *s);
HWND hTreeWndCtrl;
HTREEITEM hTreeWnd[NUM];
HTREEITEM hTreeCurrent;

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE
                   hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
 {
   MSG msg;
   HWND hWnd;
   WNDCLASS wc;
   RECT WinDim;
   HANDLE hAccel;
```

```
wc.style          = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc    = (WNDPROC)WndProc;
wc.cbClsExtra     = 0;
wc.cbWndExtra     = 0;
wc.hInstance      = 0;
wc.hIcon          = LoadIcon(hInstance, lpszAppName);
wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
wc.lpszMenuName   = lpszAppName;
wc.lpszClassName  = lpszAppName;

if(!RegisterWin95(&wc))
   return false;
hInst = hInstance; /* save the current instance handle */
hWnd = CreateWindow (lpszAppName,
                     lpszTitle,
                     WS_OVERLAPPEDWINDOW,
                     CW_USEDEFAULT, 0,
                     CW_USEDEFAULT, 0,
                     NULL,
                     NULL,
                     hInstance,
                     NULL
                    );
if(!hWnd)
   return false;
InitCommonControls();

GetClientRect(hWnd, &WinDim);  /* get size of parent window */

/* create a tree view */
hTreeWndCtrl = CreateWindow(
               WC_TREEVIEW,
               "",
               WS_VISIBLE | WS_TABSTOP | WS_CHILD |
               TVS_HASLINES | TVS_HASBUTTONS |
               TVS_LINESATROOT,
               0, 0, 100, 100,
               hWnd,
               NULL,
               hInst,
               NULL
          );
InitTree();
```

```
   /* load accelerators */
   hAccel = LoadAccelerators(hInstance, "MYMENU");

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
   while(GetMessage(&msg, NULL, 0,0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
   return(msg.wParam);
 }

BOOL RegisterWin95(CONST WNDCLASS* lpwc)
 {
   WNDCLASSEX wcex;

   wcex.style           = lpwc->style;
   wcex.lpfnWndProc     = lpwc->lpfnWndProc;
   wcex.cbClsExtra      = lpwc->cbClsExtra;
   wcex.cbWndExtra      = lpwc->cbWndExtra;
   wcex.hInstance       = lpwc->hInstance;
   wcex.hIcon           = lpwc->hIcon;
   wcex.hCursor         = lpwc->hCursor;
   wcex.hbrBackground   = lpwc->hbrBackground;
   wcex.lpszMenuName    = lpwc->lpszMenuName;
   wcex.lpszClassName   = lpwc->lpszClassName;
   wcex.cbSize          = sizeof(WNDCLASSEX);
   wcex.hIconSm         = LoadIcon(wcex.hInstance, "SMALL");
   return RegisterClassEx(&wcex);
 }

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam,
                          LPARAM lParam)
 {
  HDC hdc;
  static char selection[80] = "";
  NMHDR *nmptr;
  PAINTSTRUCT paintstruct;
  int i;
```

```
switch(uMsg)
 {
   case WM_COMMAND:
      switch(LOWORD(wParam))
       {
         case IDM_EXPAND:
            TreeView_Expand(hTreeWndCtrl, hTreeCurrent,
                            TVE_EXPAND);
            break;
         case IDM_EXPANDALL:
            for(i=0; i<NUM; i++)
               TreeView_Expand(hTreeWndCtrl, hTreeWnd[i],
                               TVE_EXPAND);
            break;
         case IDM_COLLAPSE:
            TreeView_Expand(hTreeWndCtrl, hTreeCurrent,
                            TVE_COLLAPSE);
            break;
         case IDM_TEST :
            break;
         case IDM_EXIT :
            DestroyWindow(hWnd);
            break;
       }
      break;
 case WM_NOTIFY:
   nmptr = (LPNMHDR) lParam;
   if(nmptr->code == TVN_SELCHANGED) {
     InvalidateRect(hWnd, NULL, 1);
     if(((LPNM_TREEVIEW)nmptr)->itemNew.hItem == hTreeWnd[0])
        strcpy(selection, "One.");
     else if(((LPNM_TREEVIEW)nmptr)->itemNew.hItem ==
            hTreeWnd[1])strcpy(selection, "Two.");
     if(((LPNM_TREEVIEW)nmptr)->itemNew.hItem == hTreeWnd[2])
        strcpy(selection, "Three.");
     if(((LPNM_TREEVIEW)nmptr)->itemNew.hItem == hTreeWnd[3])
        strcpy(selection, "Four.");
     if(((LPNM_TREEVIEW)nmptr)->itemNew.hItem == hTreeWnd[4])
        strcpy(selection, "Five.");
     hTreeCurrent = ((LPNM_TREEVIEW)nmptr)->itemNew.hItem;
   }
   break;
```

```
      case WM_PAINT:
         hdc = BeginPaint(hWnd, &paintstruct);
         report(hdc, selection);
         EndPaint(hWnd, &paintstruct);
         break;
      case WM_DESTROY :
         PostQuitMessage(0);
         break;
      default:
         return (DefWindowProc(hWnd, uMsg, wParam, lParam));
    }
   return(0L);
 }

/* Initialize the tree list. */
void InitTree(void)
{
  TV_INSERTSTRUCT tvs;
  TV_ITEM tvi;

  tvs.hInsertAfter = TVI_LAST; /* make tree in order given */
  tvi.mask = TVIF_TEXT;

  tvi.pszText = "One";
  tvs.hParent = TVI_ROOT;
  tvs.item = tvi;
  hTreeWnd[0] = TreeView_InsertItem(hTreeWndCtrl, &tvs);
  hTreeCurrent = hTreeWnd[0];

  tvi.pszText = "Two";
  tvs.hParent = hTreeWnd[0];
  tvs.item = tvi;
  hTreeWnd[1] = TreeView_InsertItem(hTreeWndCtrl, &tvs);

  tvi.pszText = "Three";
  tvs.item = tvi;
  tvs.hParent = hTreeWnd[1];
  hTreeWnd[2] = TreeView_InsertItem(hTreeWndCtrl, &tvs);
```

```
  tvi.pszText = "Four";
  tvs.item = tvi;
  tvs.hParent = hTreeWnd[2];
  hTreeWnd[3] = TreeView_InsertItem(hTreeWndCtrl, &tvs);

  tvi.pszText = "Five";
  tvs.item = tvi;
  tvs.hParent = hTreeWnd[2];
  hTreeWnd[4] = TreeView_InsertItem(hTreeWndCtrl, &tvs);
}

/* Report Selection */
void report(HDC hdc, char *s)
{
  char str[80];

  if(*s) {
    strcpy(str, "Selection is ");
    strcat(str, s);
  }
  else strcpy(str, "No selection has been made.");
  TextOut(hdc, 0, 200, str, strlen(str));
}
```

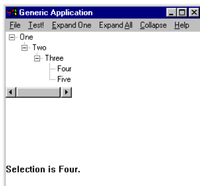
התוכנית זקוקה לקובץ המשאבים הבא:

```
#include <windows.h>
#include "tree.h"

MYMENU MENU
{
  MENUITEM "&Expand One", IDM_EXPAND
  MENUITEM "Expand &All", IDM_EXPANDALL
  MENUITEM "&Collapse", IDM_COLLAPSE
  MENUITEM "&Help", IDM_HELP
}
```

להלן קובץ הכותר tree.h:

```
#define IDM_EXPAND    100
#define IDM_EXPANDALL 101
#define IDM_COLLAPSE  102
#define IDM_HELP      103
```



תרשים 14.4: דוגמת פלט של פקד תצוגת העץ

הפונקציה ()InitTree מאתחלת את פקד תצוגת העץ. שים לב שהידיות של כל הפריטים נמצאות במערך hTreeWnd. ידיות אלו משמשות לזיהוי פריטים בעת בחירתם מרשימת העץ. hTreeCurrent מזהה את הפריט הנוכחי שנבחר. ידית זו נועדה למקרה שהמשתמש מרחיב, או מכווץ ענף כלשהו בעזרת התפריט.

הפונקציה ()WndProc מטפלת בהודעות WM_NOTIFY שמתקבלות בעת בחירת פריטים חדשים. ערכו של itemNew נבדק כנגד רשימת הידיות של הפריטים שבמערך hTreeWnd. כאשר נמצאת ידית תואמת, מדווחת הפונקציה על הבחירה החדשה.

הדוגמה שהבאנו ממחישה את ההיבטים הבסיסיים של פקדי תצוגת העץ, ומציגה את הנושא באופן כללי ביותר. לדוגמה, תצוגת העץ מאפשרת לגרור פריטים מעץ אחד ולשחרר אותם בעץ אחר. נושא זה ואחרים תוכל לחקור בעצמך.

פרק **15**

ניהול זיכרון

15.1 מודל הזיכרון Win32

כפי שאולי הצלחת לנחש מסעיפים שכבר למדת, מודל הזיכרון של Win32 מקל מאוד על ניהול הזיכרון. המודלים **קטן** (Small), **גדול** (Large), **וענק** (Huge) אינם קיימים עוד בגרסת 32 סיביות של Windows (32-bit Windows). מכיון שאין מודלים של זיכרון, תוכניות Win32 אינן מבדילות יותר בין זיכרון **קרוב** (Near) **ורחוק** (Far). בלי **סגמנטים** (Segments), התוכנית והנתונים שלה ממוקמים באותו זיכרון ליניארי, שהופך את הטיפול בתוכניות גדולות ובקטעי נתונים לפשוט וקל יותר.

בסביבת העבודה של Win32, לכל תהליך יש מרחב כתובות וירטואליות של 32 סיביות משלו, שמגיע עד ארבעה ג'יגה-בית (4 Gigabytes ,4GB). Windows מקצה למשתמש שני ג'יגה בית בזיכרון הנמוך (0x00000000 עד 0x7FFFFFFF), ושומרת על שני ג'יגה בית בזיכרון הגבוה (0x80000000 עד 0xFFFFFFFF) עבור **גרעין המחשב** (Computer's Kernel). הכתובות שמשתמשים בהן התהליכים אינן מייצגות עוד מקומות פיסיים ממשיים בזיכרון. במקום זאת, גרעין **מערכת ההפעלה** (התוכנה הבסיסית של מערכת ההפעלה ששולטת במעבד, בזיכרון, במטלות, ועוד) מחזיק עבור כל תהליך **מפת דף** (Page Map), ששמשת לתרגום **כתובות וירטואליות** (Virtual Addresses) לכתובות פיסיות, כפי שדרוש. **תהליך** (Process) אינו יכול לכתוב מחוץ למרחב הכתובות שהוקצה לו (וכך יש הגנה מפני פגיעת תהליך אחד באחר).

Win32 API, ממשק התכנות של 32 סיביות, תומך בפונקציות ההקצאה GlobalAlloc ו-LocalAlloc. בסביבת Win32, הקצאות גלובליות ולוקליות זהות למעשה. בסביבת Win16, **הקצאה לוקלית** (Local Allocation) היתה במרחב הכתובות של התהליך **והקצאה גלובלית** (Global Allocation) היתה מחוץ למרחב הכתובות שלו. בסביבת Win32, מערכת ההפעלה מקצה את שני סוגי הזיכרון במרחב הכתובת של התהליך עצמו, וכך היא מאפשרת לגשת לשני סוגי הזיכרון ישירות מתוך התוכניות, על ידי שימוש במצביעים בני 32 סיביות. עם זאת, כמו שנלמד בהמשך, ייתכן שהקצאה לוקלית תורמת לכך שקל יותר להבין את התוכניות, ואסור לזלזל בהיבט זה.

סביבת העבודה של Win32 מציגה שתי שיטות חדשות שבהן יכולות התוכניות לנהל את הזיכרון: **מנהל זיכרון וירטואלי** (Virtual Memory Manager) ו**מנהל ערימה לוקאלי** (Local Heap Manager). פונקציות API לניהול הזיכרון הווירטואלי דומות לפונקציות API המטפלות בניהול הזיכרון הגלובלי, אלא שהתוכניות יכולות לקבל **מלאי** (Reserve) בלוקים גדולים של זיכרון וירטואלי ואחר כך להקצות אותם כנדרש. **מנהל הערימה** (Heap Manager) החדש שונה ממנהלי הערימה שמשתמשים בהם בדרך כלל. מנהל הערימה החדש מאפשר לתוכניות ליצור **ערימות רבות** (Multiple Heaps) וגם **ערימות נפרדות** (Separate Heaps). הערימות הרבות מאפשרות שימוש יעיל ופשוט להקצאת כמויות קטנות של זיכרון (כמו הזיכרון שדרוש למצביע בודד). בסעיפים שיבואו בהמשך, תלמד יותר על הדרך שבה Windows מנהלת את הזיכרון. כמו כן, תלמד על חלק מהפונקציות שמשתמשים בהם בתוכניות כדי לנהל בצורה יעילה את הזיכרון הגלובלי והווירטואלי של Windows.

15.2 זיכרון גלובלי וזיכרון לוקלי

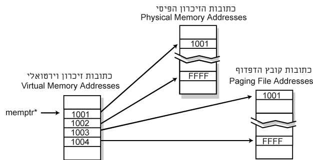
בסעיף קודם למדת שבמודל הליניארי בן 32 הסיביות של יישומי Win32, Windows (ולכן גם התוכניות שלך) אינה מבדילה בין זיכרון גלובלי לבין זיכרון לוקלי. כתוצאה מכך, Windows (ולכן גם התוכניות שלך), אינה מבדילה בין ערימה גלובלית לבין ערימה לוקלית. על כן, האובייקטים של הזיכרון שהתוכניות מקצות על ידי השימוש בפונקציות GlobalAlloc ו-LocalAlloc הינם זהים. Windows מקצה את הזיכרון כ**פרטי** (In Private), ב**דפים שמורים** (Committed Pages), כלומר: דפי זיכרון שאינם נגישים לתוכניות אחרות) עם אפשרות גישה לקריאה/כתיבה. **זיכרון פרטי** (Private Memory) הוא זיכרון שתוכניות אחרות, ואפילו תוכניות שרצות (או פועלות) כרגע, אינן יכולות לגשת אליו.

הפונקציות GlobalAlloc ו-LocalAlloc יכולות להקצות בלוק של זיכרון בכל גודל שיכול להיות מיוצג על ידי 32 סיביות ולהתאים לזיכרון הקיים, כולל מקום האחסון **בקובץ הדפדוף** (Paging File), שנלמד עליו יותר בהמשך. אובייקטי הזיכרון שמוקצים בתוכניות יכולים להיות באחד משני מצבים: **קבוע** (במקומות מסוימים בזיכרון) או **בר-העברה** (Movable). כאשר מקצים אובייקט זיכרון קבוע, הוא יישאר במקום הנתון שלו בזיכרון הפיסי למשך כל משך החיים שלו.

את האובייקטים שניתן להעביר אפשר לסמן כ**ניתנים למחיקה** (Discardable). ב-Windows 3.x, אובייקטי זיכרון שניתנים להעברה היו חשובים לניהול הזיכרון. לעומת זאת, ב-Win32, משתמשים בתוכניות בזיכרון וירטואלי, ולכן המערכת יכולה לנהל את הזיכרון מבלי שתהיה השפעה כלשהי על הכתובות הווירטואליות. כאשר המערכת מעבירה דף זיכרון, Windows ממפה את הדף הווירטואלי למקום חדש. התוכנית משתמשת בזיכרון בר-העברה כדי להקצות זיכרון שניתן למחיקה, אשר התוכנית תשתמש בו לעיתים קרובות וגם תמחק אותו מדי פעם (למשל, עבור מערכים קטנים, מצביעים, וכדומה).

15.3 הזיכרון הווירטואלי

בסעיפים קודמים הצגנו את המונח **זיכרון וירטואלי** (Virtual Memory). בסעיף הקודם הסברנו שמנהל הזיכרון הווירטואלי מאפשר לתוכניות לראות את **זיכרון המחשב** הפיסי, ואת **קובץ הדפדוף** (שהוא בדרך כלל גדול יותר) - כגוש אחד של **זיכרון רציף**. כדי לאפשר לתוכניות לנהל את הזיכרון בדרך זו, מנהל הזיכרון הווירטואלי מאפשר לתוכניות לעבוד עם **מפות** (Maps) המתייחסות לזיכרון הממשי, במקום לעבוד עם הזיכרון הממשי עצמו. מכיון שעובדים עם זיכרון רציף מדומה, ולא עם גוש זיכרון אמיתי רציף, המעצבים של Windows קראו למודל הזיכרון הזה בשם **מודל הזיכרון הווירטואלי** (Virtual Memory Model). תרשים 15.1 מציג מודל לוגי, שמראה כיצד Windows משתמשת בזיכרון וירטואלי כדי לגשת לזיכרון פיסי.



תרשים 15.1 : Windows משתמשת בזיכרון וירטואלי כדי לגשת לזיכרון פיסי.

בגלל הדרך שבה Windows מנהלת את הזיכרון הווירטואלי, מרחב הכתובות הווירטואליות של כל תהליך גדול הרבה יותר ממרחב הכתובות הפיסיות של כל התהליכים גם יחד. Windows משתמשת בכונן הקשיח כמקום אחסון נוסף, של הזיכרון הווירטואלי. כמות הזיכרון הכללי שעומדת לרשות תהליך כלשהו, הינה הסכום של הזיכרון הפיסי ושל מרחב המקום **החופשי** ב**קובץ הדפדוף** (Paging File) של Windows, שנמצא בדיסק הקשיח. קובץ הדפדוף הוא קובץ בדיסק ש-Windows משתמשת בו כדי להגדיל את הזיכרון האפקטיבי של המחשב. Windows מארגנת את מרחב הזיכרון הווירטואלי **בדפים** (Pages), או **יחידות זיכרון** (Units Of Memory). גודל הדף תלוי במחשב ה**מארח** (Host). באפשרותך לקרוא לפונקציה GetSystemInfo כדי לדעת מה גודל הדף במחשב.

למדת שבמודל הזיכרון Win32, מערכת ההפעלה מספקת לכל **תהליך** (Process) את מרחב הזיכרון הפרטי שלו. כאשר **מטלה** (Thread) מופעלת בתהליך, היא יכולה לגשת לזיכרון ששייך לתהליך שלה בלבד. הזיכרון ששייך לתהליכים האחרים נסתר מפני

המטלה הפועלת, ואין לה גישה אליו. מכיון שלכל תהליך יש מרחב זיכרון וירטואלי פרטי של 4GB, הוא יכול לראות את הזיכרון כאילו הוא נמצא במרחב הכתובות 0x00000000 עד 0xFFFFFFFF. שני תהליכים שפועלים בו-זמנית יכולים לאחסן זיכרון בכתובת 0x12341234 מבלי לפגוע זה בזה. שימוש באותה כתובת לא היה אפשרי, אם שני התהליכים היו משתמשים באותו מרחב זיכרון.

במציאות, מנהל הזיכרון הווירטואלי ממפה את הכתובת הווירטואלית לכתובת פיסית ממשית - שיכולה להיות בזיכרון הפיסי של המחשב, או בקובץ הדפדוף. מה שחשוב הוא, שהיישום יתייחס לכתובת הזיכרון 0x12341234 - ולא משנה אם היא נמצאת בזיכרון הפיסי (RAM) בכתובת ממשית 0x00012345, או בסקטור 14352 שבכונן הדיסק של המחשב. לכן, מנהל הזיכרון הווירטואלי מאפשר לך להפעיל תוכניות רבות בו-זמנית, מבלי לגרום לך לבחון כל הזמן אם תוכנית אחת אינה פוגעת בזיכרון של תוכניות אחרות שפועלות במקביל במערכת.

ב- Windows, מערכת ההפעלה מחלקת את מרחב הזיכרון הווירטואלי בן ה-4KB של כל תהליך לארבע מחיצות. המחיצה הראשונה, במרחב הכתובות 0x00000000 עד 0x003FFFFF (מחיצה של 4MB בתחתית מרחב הזיכרון הווירטואלי) מיועדת לשמירת תאימות עם MS-DOS ועם Windows בגרסת 16 הסיביות. אסור שהיישומים של Win32 יקראו או יכתבו במחיצה זו. אם היישום מנסה לגשת לזיכרון זה, מערכת ההפעלה מחזירה מצביע NULL והדבר עלול לגרום לאי-יציבות (שמובילה לנפילת התוכנית, נעילת מערכת ההפעלה, וכדומה).

Windows משתמשת במחיצה שבמרחב הכתובות 0x00400000 עד 0x7FFFFFFF עבור החלק הפרטי של התהליך; זהו מרחב כתובות שאינו משותף (unshared address space). תהליכים אחרים של Win32 אינם יכולים לקרוא מזיכרון זה או לכתוב בו, או לגשת בצורה כלשהי לנתונים של תהליך אחר שמאוחסנים בחלק זה, שגודלו כ- 2GB. עבור היישומים שלך, אתה צריך להחזיק את רוב החומר של התהליך שלך בתוך אזור מוגן זה. Windows משתמשת ב- 2GB הנותרים של מרחב הזיכרון הווירטואלי של התהליך כדי לאחסן בו קבצים משותפים וקבצי מערכת ההפעלה. כאשר עובדים עם הזיכרון במרחב הכתובת של התוכניות, צריך להימנע מהגישה למרחב כתובות שמעל ל- 0x80000000, אלא אם התוכנית שלך מנסה באופן מפורש לגשת לקובץ משותף או אל קובץ מערכת.

15.4 מבט נוסף על ערימות (Heaps)

למדת בסעיף 15.3 ש-Windows מקצה מרחב כתובות וירטואלי של 4GB עבור כל תהליך מופעל. פונקציות הערימה (Heap Functions) של Win32 מאפשרות לתהליך ליצור **ערימה פרטית** (Private Heap), שהיא למעשה בלוק של דף אחד או יותר במרחב הזיכרון של התהליך. הפונקציה HeapCreate יוצרת ערימה בגודל נתון, והפונקציות HeapAlloc ו-HeapFree מקצות ומשחררות את הזיכרון שבערימה. כאשר יוצרים ערימות בתוכניות Windows, הערימה מתחילה בכתובת 0x7FFFFFFF וממשיכה לגדול, אם דרוש, בתחום ה-2MB של **הזיכרון השמור לתהליך** (Reserved Process Memory).

באפשרות האובייקטים מסוג ערימה לצמוח באופן דינמי בתוך התחום שהוגדר להם בעת שנוצרו על ידי הפונקציה HeapCreate. הגודל המקסימלי של הערימה מגדיר את מספר דפי הזיכרון שנמצא ב**מלאי הערימה** (Heap Reserves). הגודל ההתחלתי מגדיר את ההקצאה הראשונית של מספר הדפים ה**שמורים** (Committed) לקריאה/כתיבה. Windows מקצה אוטומטית דפים נוספים **מהמרחב השמור** (Reserved Space) של הערימה, כאשר הדרישות של HeapAlloc עוברות את הגודל הנוכחי של **הדפים שכבר הוקצו** (Committed Pages). כלומר, Windows מקצה דפים חדשים לשימוש התוכנית מתוך מלאי הזיכרון שלה.

לאחר ש-Windows **מקצה** (Commitţs) דפים לערימה, היא אינה משחררת דפים אלה עד שהתהליך מסתיים, או עד שהתוכנית מפרקת את הערימה עם הפונקציה DestroyHeap. מכיון שלזיכרון המוקצה בערימה על ידי התוכניות עם HeapAlloc יש מקום קבוע במרחב הזיכרון הווירטואלי של התהליך והמערכת אינה יכולה לדחוס את הערימה, צריך לכתוב את היישומים כך שיגרמו למינימום של פיצול בערימה.

הזיכרון של ערימה פרטית נגיש רק לתהליך שיצר אותה. אם תיקיית קישור דינמי (Dynamic-Link Library, בקיצור DLL) יוצרת ערימה פרטית, Windows יוצרת את הערימה הזאת במרחב הכתובת של התהליך שקרא לתיקיית הקישור הדינמי (תחת Windows 9x, מעל 0x8000000). אך רק התהליך שקרא לתיקיית הקישור הדינמי יכול לגשת אל המידע של תיקיית הקישור הדינמי - הערימה הפרטית שנוצרה. פירוש הדבר, שתהליכים רבים שמפעילים את אותה תיקיית קישור דינמי, יכולים לגרום למספר רב של ערימות פרטיות שנוצרות לתיקיה זו, אבל כל מופע של תיקיה יכול לגשת לערימה פרטית אחת בלבד.

15.5 הקצאת בלוק זיכרון מהערימה הגלובלית

למדת שבאפשרות התוכניות להשתמש במספר טכניקות כדי להקצות סוגים שונים של זיכרון בסביבת Windows 9x ו- Windows NT. אחת ההקצאות הנפוצות היא הקצאת זיכרון מהערימה הגלובאלית, אשר דומה להקצאת הזיכרון שביצעת כבר בתוכניות DOS, באמצעות פונקציות כמו malloc ו-halloc. משתמשים בפונקציה GlobalAlloc כדי להקצות זיכרון מהערימה הגלובלית. פונקציה זו מקצה מהערימה את מספר הבתים שפי שמוגדר בפרמטר שלה. כותבים את הפונקציה GlobalAlloc כמו בהגדרה שלהלן:

```
HGLOBAL GlobalAlloc (
  UINT uFlags,            // object allocation attributes
  UINT dwBytes            // number of bytes to allocate
);
```

הפרמטר uFlags מגדיר כיצד רוצים להקצות את הזיכרון. אם הפרמטר uFlags הוא אפס, ברירת המחדל היא הדגל GMEM_FIXED. מלבד הצירופים שאינם תואמים, אשר מובאים בטבלאות שלהלן, כאשר התוכנית קוראת לפונקציה GlobalAlloc באפשרותה

להשתמש בכל שילוב של דגלים שמפורטים בטבלאות. כדי לציין אם הפונקציה הקצתה זיכרון **קבוע** (Fixed) או **בר-העברה** (Movable), צריך להגדיר את אחד הדגלים שמפורטים בטבלה 15.1.

טבלה 15.1: סוגי הקצאת זיכרון לשימוש עם **GlobalAlloc**.

דגל	פירוש
GMEM_FIXED	מקצה זיכרון קבוע. התוכניות אינן יכולות לשלב דגל זה עם הדגל GMEM_MOVEABLE או GMEM_DISCARDABLE. הערך המוחזר מצביע לבלוק הזיכרון, כדי שאפשר יהיה לגשת אליו.
GMEM_MOVEABLE	מקצה זיכרון בר-העברה. התוכניות אינן יכולות לשלב דגל זה עם הדגל GMEM_FIXED. הערך המוחזר הוא ידית של אובייקט הזיכרון, המייצגת ערך בן 32 סיביות, והיא פרטית לתהליך הקורא. כדי לתרגם את הידית למצביע, צריך להשתמש בפונקציה GlobalLock.
GPTR	משלב את הדגל GMEM_FIXED עם הדגל GMEM_ZEROINIT שבטבלה 15.2.
GHND	משלב את הדגל GMEM_MOVEABLE עם הדגל GMEM_ZEROINIT שבטבלה 15.2.

בנוסף לדגלים שמצויינים בטבלה 15.1, הפרמטר uFlags יכול להכיל כל ערך מהערכים שמפורטים בטבלה 15.2, מלבד המקומות שהטבלה מציינת שאי אפשר לשלב את הדגל עם דגל אחר.

טבלה 15.2: ערכי דגלים נוספים לשימוש עם הפונקציה **GlobalAlloc**.

דגל	פירוש
GMEM_DDESHARE	מקצה זיכרון שמשמש פונקציות **חילוף נתונים דינמי** (Dynamic Data Exchange, בקיצור DDE), עבור תקשורת במסגרת חילוף נתונים דינמי. דגל זה שימושי לתאימות עם יישומי Win16. חלק מהיישומים יכולים להשתמש ב-GMEM_DDESHARE כדי לשפר את פעולות חילוף הנתונים הדינמי, ועל התוכניות להגדיר את הדגל GMEM_DDESHARE אם הן ישתמשו בזיכרון לחילוף נתונים דינמי. רק יישומים שמשתמשים בחילוף נתונים דינמי, או הלוח (Clipboard) המשמש להתקשרות בין תהליכים, צריכים להגדיר את הדגל GMEM_DDESHARE.
GMEM_DISCARDABLE	מקצה זיכרון שאפשר למחוק אותו כשאינו דרוש עוד (זהו זיכרון שאינו קבוע בכתובת ספציפית במרחב הכתובת הווירטואלית של התהליך). התוכניות אינן יכולות לחבר דגל זה עם הדגל GMEM_FIXED. ייתכן שחלק מהיישומים מבוססי Win32 יתעלמו מדגל זה.

דגל	פירוש
GMEM_LOWER	Win32 מתעלמת מדגל זה. Win32 API מספק דגל זה לתאימות עם גרסאות קודמות של Windows.
GMEM_NOCOMPACT	אינו מדוחס או מוחק זיכרון כדי לספק דרישות להקצאת זיכרון.
GMEM_NODISCARD	אינו מוחק זיכרון כדי לספק דרישות הקצאת הזיכרון.
GMEM_NOT_BANKED	Win32 מתעלמת מדגל זה. Win32 API מספק דגל זה לצורך תאימות עם גרסאות קודמות של Windows.
GMEM_NOTIFY	Win32 מתעלמת מדגל זה. Win32 API מספק דגל זה לתאימות עם גרסאות קודמות של Windows.
GMEM_SHARE	מקצה זיכרון שמשמש את פונקציות חילוף הנתונים הדינמי (Dynamic Data Exchange, בקיצור DDE), לתקשורת במסגרת חילוף נתונים דינמי. כמו הדגל GMEM_DDESHARE.
GMEM_ZEROINIT	מאתחל את הזיכרון באפס.

בנוסף להגדרת סוג הזיכרון שעל הפונקציה GlobalAlloc להקצות, התוכנית חייבת לפרט באמצעות הפרמטר dwBytes את מספר הבתים שצריך להקצות. אם פרמטר שווה לאפס, והפרמטר uFlag מגדיר את הדגל GMEM_MOVEABLE, הפונקציה תחזיר ידית לאובייקט זיכרון ש-Windows מסמנת כניתן למחיקה. אם הפונקציה מצליחה, הערך המוחזר יהיה הידית של אובייקט הזיכרון החדש שהוקצה; אם הפונקציה נכשלת, הערך המוחזר יהיה NULL.

אם הערימה אינה מכילה די שטח חופשי כדי לספק את הדרישה, GlobalAlloc מחזירה NULL. מכיון ש-GlobalAlloc משתמשת בערך NULL כדי לציין שגיאה, Windows לעולם אינה מקצה כתובת וירטואלית שערכה אפס. לכן קל לגלות את השימוש במצביע NULL. Windows יוצרת את כל הזיכרון עם גישה להפעלה, ואינה מחייבת פונקציה מיוחדת כדי להפעיל באופן דינמי קוד שנוצר. Windows מבטיחה שזיכרון שהוקצה על ידי התוכנית עם הפונקציה GlobalAlloc יקבע על פי גבולות של 8 בתים (8 Bytes Boundary).

Windows מגבילה את הפונקציות GlobalAlloc ו-LocalAlloc להקצות יחד עד 65,536 ידיות לכל תהליך, עבור זיכרון GMEM_MOVEABLE (עבור זיכרון שמוקצה גלובלית) ועבור זיכרון LMEM_MOVEABLE (עבור זיכרון שמוקצה לוקלית). הגבלה זו אינה חלה על זיכרון GMEM_FIXED או LMEM_FIXED. אם הפונקציה GlobalAlloc מצליחה, היא מקצה לפחות את כמות הזיכרון שנדרשה בקריאה לפונקציה. אם הכמות שבפועל ש-GlobalAlloc מקצה גדול יותר מהכמות שנדרשה בקריאה לפונקציה, היישום יכול להשתמש בכמות כולה. כדי לדעת את מספר הבתים שהוקצו בפועל על ידי הפונקציה GlobalAlloc, משתמשים בפונקציה GlobalSize.

כדי להבין יותר טוב את פעולת הפונקציה GlobalAlloc, התבונן בתוכנית **Global_Alloc**, שנמצאת בתקליטור המצורף לספר זה (בתיקיה Books\59285\Chap15). התוכנית מקצה זיכרון כדי לאחסן מחרוזת. כאשר מתחיל היישום, הקוד שמטפל בהודעה WM_CREATE יוצר מאגר בן 27 תווים (26 בתים וה-NULL המסיים). כאשר המשתמש בוחר !Test, התוכנית מציגה את המאגר ואת תוכנו על המסך.

15.6 שימוש ב-GlobalReAlloc כדי לשנות גודל ערימה באופן דינמי

למדת בסעיף 15.5 שבאפשרות התוכניות להשתמש בפונקציה GlobalAlloc כדי להקצות זיכרון בערימה הגלובלית. אך פעמים רבות, התוכנית חייבת להקצות בלוק זיכרון מחדש, לאחר ההקצאה הראשונה. אפשר לעשות זאת על ידי הפונקציה GlobalReAlloc. פונקציה זו משנה את הגודל או המאפיינים של אובייקט זיכרון גלובלי מוגדר. הגודל יכול לגדול או לקטון, בהתאם לקריאה. את הפונקציה GlobalReAlloc כותבים בתוכניות, כמו בהגדרה זו:

```
HGLOBAL GlobalReAlloc (
  HGLOBAL hMem,          // handle to the global memory object
  DWORD dwBytes,         // new size of the block
  UINT uFlags            // how to reallocate object
);
```

הידית hMem מזהה את אובייקט הזיכרון הגלובלי ש-GlobalReAlloc מקצה מחדש. אחת משתי הפונקציות GlobalAlloc או GlobalReAlloc החזירו ידית זאת קודם. הפרמטר dwBytes מגדיר את הגודל החדש, בבתים של בלוק הזיכרון. אם הפרמטר dwBytes שווה לאפס והפרמטר uFlags מגדיר את הדגל GMEM_MOVEABLE, הפונקציה מחזירה את הידית של אובייקט זיכרון ש-Windows סימנה כניתן למחיקה. אם הפרמטר dwBytes שווה לאפס והפרמטר uFlags מגדיר את הדגל GMEM_MODIFY, פונקציית API מתעלמת מהפרמטר dwBytes. הפרמטר uFlags מגדיר כיצד להקצות מחדש את אובייקט הזיכרון הגלובלי. אם הפרמטר uFlags מגדיר את הדגל GMEM_MODIFY, הפרמטר uFlags מגדיר את המאפיינים של אובייקט הזיכרון, ופונקציית API מתעלמת מהפרמטר dwBytes; אחרת, הפרמטר uFlags שולט בהקצאה מחדש של אובייקט הזיכרון.

כאשר קוראים לפונקציה GlobalReAlloc, התוכנית יכולה לשלב את הדגל GMEM_MODIFY עם אחד או שני הדגלים שמפורטים בטבלה 15.3.

טבלה 15.3: דגלים שמתאימים לשימוש עם הדגל **GMEM_MODIFY**.

דגל	פירוש
GMEM_DISCARDABLE	מקצה זיכרון שניתן למחיקה, כאשר מגדירים גם את הדגל GMEM_MODIFY. Windows מתעלמת מדגל זה, אם האובייקט הוקצה מקודם כ**בר-העברה** (Movable), או אם הוגדר הדגל GMEM_MOVEABLE.
GMEM_MOVEABLE	עבור Windows NT בלבד: משנה אובייקט זיכרון קבוע לאובייקט זיכרון בר-העברה, כאשר מגדירים את הדגל GMEM_MODIFY.

אם הפרמטר uFlags אינו מגדיר את GMEM_MODIFY, הוא יכול להיות כל שילוב של הדגלים שמפורטים בטבלה 15.4.

טבלה 15.4: דגלים נוספים עבור הפרמטר **uFlags**.

דגל	פירוש
GMEM_MOVEABLE	כאשר dwBytes שווה לאפס, GMEM_MOVEABLE מוחק את בלוק הזיכרון הקודם בר-ההעברה ואשר ניתן למחיקה. כאשר **מונה הנעילה** (Lock Count) של האובייקט אינו שווה לאפס, או אם הוא אינו בר-העברה וניתן למחיקה, הפונקציה נכשלת. כאשר dwBytes אינו שווה לאפס, GMEM_MOVEABLE יאפשר למערכת להעביר את הבלוק שהוקצה שוב למקום חדש, מבלי לשנות את המאפיינים **בר-העברה** (Movable) או **קבוע** (Fixed) של אובייקט הזיכרון. אם האובייקט קבוע, יכול להיות שהידית אשר הפונקציה מחזירה תהיה שונה מהידית המוגדרת על ידי הפרמטר hMem. כאשר האובייקט בר-העברה, התוכנית יכולה להעביר את הבלוק מבלי לגרום לידית האובייקט להיות בלתי-תקפה (Invalidating The Handle), אפילו אם קריאה קודמת לפונקציה GlobalLock נועלת את האובייקט כרגע. כדי להשיג את הכתובת החדשה של בלוק הזיכרון, צריך להשתמש ב-GlobalLock.
GMEM_NOCOMACT	מונע מ-Windows מלדחוס או למחוק זיכרון לצורך סיפוק דרישת הקצאת הזיכרון.
GMEM_ZEROINIT	גורם ל-Windows לאתחל את תוכן הזיכרון הנוסף באפס, כאשר אובייקט הזיכרון גדל.

אם הפונקציה מצליחה, הערך המוחזר הוא הידית של אובייקט הזיכרון שהוקצה מחדש; אם הפונקציה נכשלת, הערך המוחזר הוא NULL. אם GlobalReAlloc מקצה מחדש אובייקט בר-העברה, הערך המוחזר הוא ידית של אובייקט הזיכרון. כדי להמיר את הידית למצביע, צריך להשתמש בפונקציה GlobalLock. אם GlobalReAlloc מקצה מחדש אובייקט קבוע, הערך המוחזר של הידית יהיה הכתובת של הבית הראשון של בלוק הזיכרון. כדי לגשת לזיכרון, באפשרות התהליך פשוט להשתמש ב**המרת סוג** (Cast) של הערך המוחזר למצביע. אם GlobalReAlloc נכשלת, היא אינה משחררת את הזיכרון המקורי, והידית והמצביע המקוריים יישארו תקפים.

כדי להבין טוב יותר את פעולת הפונקציה GlobalReAlloc, התבונן בתוכנית **Global_ReAlloc** שנמצאת בתקליטור המצורף לספר זה (בתיקיה Chap15). התוכנית פועלת במידה רבה כמו התוכנית **Global_Alloc**. עם זאת, התוכנית **Global_ReAlloc** גם מקצה מחדש עוד 27 בתים של זיכרון כדי להציג את האלף-בית באותיות קטנות כאשר המשתמש בוחר באפשרות !Test.

15.7 מחיקת בלוק זיכרון שהוקצה

בוודאי השתמשת בעבר בפונקציות **free** ו-**delete** כדי למחוק (Discard) זיכרון שהוקצה בתוכניות. כאשר מקצים או מקצים מחדש זיכרון מהערימה הגלובלית, צריך להשתמש בפונקציה GlobalDiscard כדי לשחרר את הזיכרון לאחר שהתוכנית סיימה את השימוש בו. הפונקציה GlobalDiscard מוחקת בלוק זיכרון גלובלי שהוקצה קודם על ידי השימוש בדגל GMEM_DISCARDABLE. מונה הנעילה של אובייקט הזיכרון שרוצים למחוק חייב להיות אפס, או שהפונקציה תיכשל במחיקת הזיכרון. משתמשים בפונקציה GlobalDiscard בתוכניות כמו בהגדרה שלהלן:

```
HGLOBAL GlobalDiscard(
  HGLOBAL hglbMem       // handle to the global memory object
);
```

הפרמטר hglbMem מזהה את אובייקט הזיכרון הגלובלי שרוצים למחוק. אם הפונקציה מצליחה, היא מחזירה את הידית של אובייקט הזיכרון (כלומר, הידית hglbMem). אם הפונקציה נכשלת, היא מחזירה את הערך NULL.

הפונקציה GlobalDiscard מוחקת רק אובייקטים גלובליים שהתהליך הקורא הקצה על ידי השימוש בדגל GMEM_DISCARDABLE. אם התהליך מנסה למחוק אובייקט קבוע או נעול, הפונקציה נכשלת. למרות שהפונקציה GlobalDiscard מוחקת את בלוק הזיכרון של האובייקט, ידית האובייקט נשארת תקפה. התהליך יכול להעביר באופן עקיף את הידית אל הפונקציה GlobalReAlloc כדי להקצות בלוק זיכרון גלובלי אחר שמוגדר על ידי אותה ידית.

כדי להבין טוב יותר את פעולת הפונקציה GlobalDiscard, התבונן בתוכנית **Global_Discard** שנמצאת בתקליטור המצורף (בתיקיה Chap15).

15.8 הפונקציה GlobalFree

בסעיף הקודם למדת על הפונקציה GlobalDiscard, אשר התוכניות משתמשות בה, כדי למחוק בלוקים של זיכרון שהוקצו קודם, ולהשאיר את ידית הזיכרון שנמחק לשימוש בעתיד. עם זאת, כאשר יודעים שהתוכנית לא תשתמש יותר באותו בלוק זיכרון, ואם רוצים לשמור על התוכנית מלהשתמש באותו בלוק זיכרון, או אם לא בטוחים אם התוכנית הקצתה את הזיכרון על ידי השימוש בדגל GMEM_DISCARDABLE, התוכנית יכולה להשתמש בפונקציה GlobalFree כדי לשחרר אובייקט זיכרון. הפונקציה GlobalFree משחררת את אובייקט הזיכרון הגלובלי המוגדר וגורמת לידית האובייקט של הזיכרון להיות לא-תקפה. משתמשים בפונקציה GlobalFree בתוכניות כמו בהגדרה שלהלן:

```
HGLOBAL GlobalFree(HGLOBAL hMem);
```

הפרמטר hMem מציין את אובייקט הזיכרון הגלובלי שרוצים לשחרר. כאן, לא כמו בפונקציה GlobalDiscard, כאשר הפונקציה GlobalFree מצליחה, הערך המוחזר הוא NULL. אם הפונקציה GlobalFree נכשלת, הערך המוחזר יהיה שווה לידית אובייקט הזיכרון הגלובלי.

חריגה מסוג **הרס ערימה** (Heap Corruption) או **שגיאת גישה** (Access Violation) - EXCEPTION_ACCESS_VIOLATION - יכולה לקרות כאשר התהליך מנסה לבחון את הזיכרון או לשנות אותו, לאחר שכבר שחרר את הזיכרון קודם לכן. אם הפרמטר hglbMem שווה ל-NULL, GlobalFree נכשלת והמערכת תיצור חריגה מסוג שגיאת גישה. שתי הפונקציות GlobalFree ו-LocalFree משחררות **אובייקט זיכרון נעול** (Locked-Memory Object). לאובייקט זיכרון נעול יש מונה נעילה גדול מאפס. הפונקציה GlobalLock נועלת את אובייקט זיכרון גלובלי (המונע מפונקציה אחרת למחוק את אובייקט הזיכרון) ומגדילה את מונה הנעילה של האובייקט באחד. הפונקציה GlobalUnlock מבטלת את הנעילה של אובייקט הזיכרון ומפחיתה את מונה הנעילה באחד. לקבלת מונה הנעילה של אובייקט זיכרון גלובלי, צריך להשתמש בפונקציה GlobalFlag.

✍ **הערה:** תחת Windows NT, אם היישום מופעל תחת מנפה שגיאות גירסה (DBG) של Windows NT, כמו שמופץ בתקליטור SDK CD_ROM, הפונקציות GlobalFree ו-LocalFree מכניסות נקודת עצירה לפני שחרור האובייקט הנעול. דבר זה מאפשר למתכנת לבדוק בדיקה כפולה של ההתנהגות הדרושה. כאשר במצב זה מקלידים G בזמן השימוש במנפה השגיאות (Debugger), תתבצע פעולת השחרור.

15.9 הפונקציות GlobalLock ו-GlobalHandle

כאמור, באפשרות התוכניות להשתמש בפונקציות GlobalAlloc ו-GlobalReAlloc כדי להקצות זיכרון מהערימה הגלובלית. אך כמו שראית, שתי פונקציות ההקצאה מחזירות ידית אל הזיכרון המוקצה. עם זאת, תרצה שרוב התוכניות תשתמשנה בזיכרון עם מצביע. באפשרותך להשתמש בפונקציות GlobalAlloc ו-GlobalReAlloc כדי להמיר בקלות את הזיכרון שהוקצה למצביע וחזרה שוב אל הידית. הפונקציה GlobalLock נועלת אובייקט זיכרון גלובלי ומחזירה מצביע אל הבית הראשון בבלוק הזיכרון של האובייקט. אין באפשרות התוכניות להעביר או למחוק את בלוק הזיכרון הקשור עם אובייקט זיכרון נעול. עבור אובייקטים של זיכרון המוקצים על ידי השימוש בדגל GMEM_MOVEABLE, הפונקציה מגדילה את מונה הנעילה שקשור עם אובייקט הזיכרון. את הפונקציה GlobalLock כותבים כפי שמוצג להלן:

```
LPVOID GlobalLock(
  HGLOBAL hMem          // address of the global memory object
);
```

הפרמטר hMem שבפונקציה GlobalLock מזהה את אובייקט הזיכרון הגלובלי. אחת משתי הפונקציות GlobalAlloc או GlobalReAlloc מחזירות את הידית הזאת. אם הפונקציה GlobalAlloc מצליחה, הערך המוחזר יהיה מצביע לבית הראשון בבלוק הזיכרון; אם היא נכשלת, הערך המוחזר הוא NULL.

מבנה הנתונים הפנימי של כל אובייקט זיכרון מכיל מונה נעילה שמאותחל באפס. עבור אובייקטים של זיכרון מסוג **בר-העברה** (Movable), GlobalLock מגדילה את המונה באחד, והפונקציה GlobalUnlock מפחיתה את המונה באחד. עבור כל קריאה שהתהליך מבצע ל-GlobalLock עבור אובייקט, התהליך חייב לבסוף לקרוא ל-GlobalUnlock. התוכנית אינה יכולה להעביר או למחוק זיכרון נעול, אלא אם כן התוכנית מקצה מחדש את אובייקט הזיכרון על ידי שימוש בפונקציה GlobalReAlloc. בלוק הזיכרון של אובייקטים נעולים נשאר נעול, עד שהתוכנית מורידה את מונה הנעילה של בלוקי הזיכרון לאפס, ואז באפשרות התוכנית להעביר, או למחוק את הזיכרון.

לאובייקטים של זיכרון שהוקצו על ידי השימוש בדגל GMEM_FIXED יש תמיד מונה נעילה שערכו אפס. עבור אובייקטים אלה, ערך המצביע המוחזר שווה לערך הידית המוגדרת. אם התוכנית מחקה כבר את בלוק הזיכרון המוגדר, או אם גודל בלוק הזיכרון שווה לאפס בתים, הפונקציה GlobalLock מחזירה NULL. ערך מונה הנעילה של האובייקטים הנמחקים הוא תמיד אפס. את הפונקציה GlobalLock כותבים בתוכניות כמו בקטע הקוד שלהלן:

```
  HGLOBAL hMem = GlobalAlloc(GHND, 27);
  LPCTSTR   pCur;
  if (hMem && (pMem = (LPTSTR) GlobalLock(hMem)) != NULL)
```

בקטע קוד זה מקצים ידית ל-27 בתים של זיכרון (המשתנה hMem), אחר כך נועלים את הזיכרון הזה במצביע pMem והדבר גורם באותו זמן להמרה לסוג מחרוזת.

כמו שלמדת, התוכניות משתמשות פעמים רבות ב-GlobalLock כדי להמיר ידיות זיכרון למצביעים. בדרך כלל, משתמשים בפונקציה GlobalHandle כדי להמיר מצביע לידית זיכרון. הסיבה העיקרית להמרת המצביע חזרה היא ההכנה לפקודה GlobalFree. את הפונקציה GlobalHandle כותבים בתוכניות כמו בהגדרה זו:

```
HGLOBAL GlobalHandle(
  LPCVOID pMem   // pointer to the global memory block
);
```

הפרמטר pMem הוא מצביע לבית הראשון של בלוק הזיכרון הגלובלי. הפונקציה GlobalLock מחזירה מצביע זה. אם הפונקציה GlobalHandle מצליחה, הערך המוחזר הוא, ידית של אובייקט הזיכרון הגלובלי המוגדר; אם הפונקציה GlobalHandle נכשלת, הערך המוחזר הוא NULL.

כאשר הפונקציה GlobalAlloc מקצה אובייקט זיכרון על ידי השימוש בדגל GMEM_MOVEABLE, היא מחזירה את ידית האובייקט. הפונקציה GlobalLock ממירה ידית זו למצביע אל בלוק הזיכרון, ו-GlobalHandle ממירה את המצביע בחזרה לידית. בדרך כלל, כותבים את הפונקציה GlobalHandle כמו שרואים בקטע הקוד שלהלן:

```
HGLOBAL hMem = GlobalHandle(pMem);

GlobalUnlock(hMem);
pMem = NULL;
hMem = GlobalReAlloc(hMem, (26*2)+1, GMEM_MOVEABLE);
```

במקרה המסוים הזה, התוכנית יוצרת את הידית, ואחר כך מבטלת את הנעילה שלה (שחרור הזיכרון). אחר כך, התוכנית מקצה מחדש את הזיכרון, כמו שנדרש. לעיתים תחזור ותמיר את הידית שוב למצביע.

15.10 בדיקת זיכרון המחשב

למדת ש-Windows מחזיקה מידע אודות הזיכרון הפיסי ואודות הזיכרון הווירטואלי של המחשב. פעמים רבות, התוכניות תדרושנה מידע על גודל הזיכרון החופשי, אשר התוכניות (היישומים) יכולות לגשת אליו. אפשרות להשתמש בפונקציה GlobalMemoryStatus כדי להשיג מידע על המצב הנוכחי של זיכרון המחשב. בסיום פעולתה, הפונקציה מחזירה מידע על הזיכרון הפיסי והווירטואלי. משתמשים בפונקציה GlobalMemoryStatus בתוכניות כמו שרואים בהגדרה שלהלן:

```
VOID GlobalMemoryStatus(
  LPMEMORYSTATUS lpBuffer   // pointer to the memory
                            // structure status
  );
```

הפרמטר lpBuffer מצביע למבנה LPMEMORYSTSTUS אשר מכיל את המידע אודות הזיכרון התקף שמוחזר על ידי GlobalMemoryStatus. לפני הקריאה לפונקציה GlobalMemoryStatus, התהליך הקורא צריך לקבוע ערך לאיבר dwLength של מבנה זה. המבנה MEMORYSTSTUS מכיל מידע על הזיכרון התקף כרגע. ממשק התכנות Windows API מגדיר את המבנה MEMORYSTSTUS, כמו בדוגמה זו:

```
typedef struct _MEMORYSTATUS {
    DWORD dwLength;          // sizeof (MEMORYSTATUS)
    DWORD dwMemoryLoad;      // percent of memory in use
    DWORD dwTotalPhys;       // bytes of physical memory
    DWORD dwAvailPhys;       // free physical memory bytes
    DWORD dwTotalPageFile;   // bytes of paging file
    DWORD dwAvailPageFile;   // free bytes of paging file
    DWORD dwTotalVirtual;    // user bytes of address space
    DWORD dwAvailVirtual;    // free user bytes
} MEMORYSTATUS;
```

כמו שאפשר לראות, המבנה MEMORYSTSTUS מאחסן מידע חשוב אודות זיכרון המחשב הזמין כרגע. טבלה 15.5 מסבירה את איברי המבנה MEMORYSTSTUS.

באפשרות היישום להשתמש בפונקציה GlobalMemoryStatus כדי לקבוע כמה זיכרון ניתן להקצות ליישום, מבלי להתנגש עם יישומים אחרים. המידע שמחזירה הפונקציה GlobalMemoryStatus משתנה במהירות, ואין הבטחה לכך ששתי קריאות עוקבות לפונקציה זו יחזירו את אותו המידע.

טבלה 15.5: איברי המבנה MEMORYSTSTUS.

איבר	תיאור
dwLength	מציין את גודל המבנה. על התהליך הקורא לקבוע ערך לאיבר זה לפני הקריאה ל-GlobalMemoryStatus.
dwMemoryLoad	מגדיר מספר בין 0 ל-100 שנותן מידע כללי אודות הניצול הנוכחי של הזיכרון. הערך 0 מציין שהזיכרון אינו בשימוש והערך 100 מציין שימוש מלא.
dwTotalPhys	מציין את מספר הבתים הכולל של הזיכרון הפיסי.
dwAvailPhys	מציין את מספר הבתים הכולל של הזיכרון הפיסי הזמין.
dwTotalPageFile	מציין את מספר הבתים הכולל שכל התוכניות יכולות לאחסן בקובץ הדפדוף (Page File). שים לב, מספר זה אינו מייצג את הגודל הפיסי הממשי של קובץ הדפדוף בדיסק.
dwAvailPageFile	ציין את מספר הבתים הזמין בקובץ הדפדוף.

איבר	תיאור
dwTotalVirtual	מציין את מספר הבתים הכולל ש-Windows יכולה לתאר באזור המשתמש של מרחב הזיכרון הווירטואלי של התהליך הקורא.
dwAvailVirtual	מציין את מספר הבתים בזיכרון שאינם במלאי השמור (Unreserved) ושאינם מוקצים (Uncommitted) באזור המשתמש שבמרחב הזיכרון הווירטואלי של התהליך הקורא.

כדי להבין יותר טוב את השימוש בפונקציה GlobalMemoryStatus, התבונן בתוכנית **Global_Mem_Status**, שנמצאת בתקליטור המצורף לספר זה. התוכנית בודקת את מצב הזיכרון הנוכחי ומחזירה את גודל הזיכרון אל חלון התוכנית.

15.11 יצירת ערימה בתוך תהליך

בסעיפים קודמים למדת להקצות זיכרון לתוכניות מהערימה הגלובלית. אך כמו שלמדת קודם, התוכניות מקצות גם בלוקים קטנים יותר של זיכרון מערימה לוקלית (פרטית). הפונקציה HeapCreate יוצרת אובייקט ערימה שהתהליך הקורא יכול להשתמש בו. הפונקציה שומרת מלאי (Reserves) של בלוק רציף במרחב הזיכרון הווירטואלי של התהליך, ומקצה מקום אחסון וירטואלי ראשוני מוגדר מבלוק המלאי. את הפונקציה HeapCreate כותבים בתוכנית, כמו בהגדרה שלהלן:

```
HANDLE HeapCreate(
  DWORD flOptions,          // heap allocation flag
  DWORD dwInitialSize,      // initial heap
  DWORD dwMaximumSize       // maximum heap size
);
```

הפרמטר flOptions מגדיר מאפייני רשות (דגלים) לערימה החדשה. דגלים אלה ישפיעו על הגישות הבאות לערימה החדשה, באמצעות קריאות לפונקציות הערימה (HeapAlloc, HeapFree, HeapReAlloc, ו-HeapSize). באפשרותך להגדיר דגל אחד או יותר מהדגלים שמפורטים בטבלה 15.6.

טבלה 15.6: הדגלים האפשריים עבור הפרמטר flOptions.

דגל	תיאור
HEAP_GENERATE_EXCEPTION	מגדיר שהמערכת תיצור הודעת חריגה, כדי לציין כישלון של פונקציה. לדוגמה, היא תציין שאין מספיק זיכרון (Out-Of-Memory), במקום להחזיר NULL.

דגל	תיאור
HEAP_NO_SERIALIZE	מגדיר שהערימה לא תשתמש בפעולה הדדית מסוג mutual exclusion, כאשר פונקציה הערימה מקצות ומשחררות זיכרון מהערימה. ברירת המחדל, כאשר לא מגדירים את הדגל HEAP_NO_SERIALIZE, היא להפעיל גישה סדרתית אל הערימה. הסדרת הגישה לערימה (Serialization Of Heap Access) מאפשרת לשתי מטלות או יותר להקצות ולשחרר זיכרון מאותה ערימה בו-זמנית.

הפרמטר dwInitialSize מגדיר את הגודל ההתחלתי של הערימה בבתים. ערך זה קובע את הכמות ההתחלתית של האחסון הפיסי ש-HeapCreate מקצה לערימה. HeapCreate מעגלת את הערך כלפי מעלה, לגבול הדף הבא. כדי לדעת את גודל הדף במחשב המארח צריך להשתמש בפונקציה GetSystemInfo. אם הפרמטר dwMaximumSize אינו אפס, הוא מציין את הגודל המקסימלי, בבתים, של הערימה. הפונקציה HeapCreate מעגלת את הערך dwMaximumSize כלפי מעלה אל גבול הדף הבא, ושומרת בלוק מלאי בגודל הזה במרחב הזיכרון הווירטואלי של התהליך עבור הערימה. אם הפונקציות HeapAlloc או HeapReAlloc מבצעות דרישות להקצאות שעוברות את גודלה של כמות האחסון הפיסי ההתחלתית כפי שהוגדרה על ידי dwInitialSize, המערכת מקצה דפים נוספים של אחסון פיסי עבור הערימה, עד לגודל המקסימלי של הערימה.

בנוסף, אם dwMaximumSize אינו שווה לאפס, הערימה לא תהיה מסוגלת לצמוח, ומתקבלת הגבלה אבסולוטית: הגודל המקסימלי של בלוק זיכרון בערימה יהיה מעט קטן יותר מ- 0x0007FFF8 בתים (גודל מרחב הכתובת הפרטי של התהליך). דרישות להקצאת בלוקים גדולים יותר ייכשל, אפילו אם הגודל המקסימלי של הערימה מספיק גדול כדי להכיל את הבלוק. אם dwMaximumSize הוא אפס, הוא מציין שהערימה יכולה לצמוח. רק שטח הזיכרון הזמין מגביל את גודל הערימה. דרישות להקצאת בלוקים גדולים מ- 0x0007FFF8 בתים אינן נכשלות אוטומטית; המערכת קוראת ל-VirtualAlloc כדי להשיג את הזיכרון הדרוש לבלוקים בגודל כזה. יישומים החייבים להקצות בלוקים גדולים של זיכרון צריכים לקבוע לפרמטר dwMaximumSize את הערך אפס.

כאשר הפונקציה מצליחה, הערך המוחזר הוא ידית לערימה החדשה שנוצרה; אם הפונקציה נכשלת, הערך המוחזר הוא NULL. כדי להשיג מידע מקיף יותר אודות השגיאה, עליך לקרוא ל-GetLastError.

הפונקציה HeapCreate יוצרת אובייקט ערימה פרטי, שהתהליך הקורא יכול להקצות ממנו בלוקים של זיכרון, על ידי שימוש בפונקציה HeapAlloc. הערך ההתחלתי שיהיה יקבע את מספר הדפים השמורים (Committed Pages) שיהיו בהקצאה הראשונית של HeapCreate עבור הערימה. הגודל המקסימלי קובע את מספר הדפים שבמלאי (Reserved Pages). דפים שמורים אלה, יחד עם הדפים שבמלאי, יוצרים בלוק רציף

במרחב הזיכרון הווירטואלי של התהליך שבו יכולה הערימה לצמוח. אם HeapAlloc מבצעת דרישות שחורגות ממספר הדפים השמורים, Windows מקצה אוטומטית דפים נוספים מלאי הדפים השמורים, בהנחה שאמצעי האחסון הפיסי זמין.

רק התהליך שיצר את אובייקט הערימה הפרטי יכול לגשת לזיכרון שמאוחסן בערימה הפרטית. אם תיקיית קישור דינמי (DLL) יוצרת ערימה פרטית, היא עושה זאת במרחב הזיכרון של התהליך שקרא לה. יותר מכך, הערימה נגישה רק לאותו תהליך. המערכת משתמשת בזיכרון מהערימה הפרטית, כדי לאחסן **מבני תמיכה בערימה** (Heap Support Structures); לכן, לא כל הגודל של הערימה זמין לתהליך. לדוגמה, אם הפונקציה HeapAlloc דורשת 64KB מערימה שהגודל המקסימלי שלה הוא 64KB, הדרישה יכולה להיכשל בגלל תקורת המערכת.

אם לא מגדירים את הדגל HEAP_NO_SERIALIZE (ברירת המחדל הפשוטה), הערימה תסדיר את הגישה בתוך התהליך הקורא. הסדרת הגישה (Serialization) מבטיחה שלא תהיה התנגשות בשעה ששתי מטלות או יותר מנסות להקצות או לשחרר זיכרון בו-זמנית מאותה ערימה.

קביעת הדגל HEAP_NO_SERIALIZE גורמת לביטול חסימת ההתנגשות ההדדית בערימה. ללא הסדרת הגישה, שתי מטלות או יותר שמשתמשות באותה ידית של הערימה, עשויות להקצות או לשחרר זיכרון בו-זמנית, דבר שעלול לפגום בערימה.

לכן, באפשרותך להשתמש בצורה בטוחה בדגל HEAP_NO_SERIALIZE רק במצבים הבאים:

- כאשר לתהליך יש מטלה אחת בלבד.
- כאשר יש לתהליך מטלות רבות, אך רק מטלה אחת קוראת לפונקציות הערימה עבור ערימה מסוימת.
- כאשר יש לתהליך מטלות רבות, והיישום מספק מנגנון משלו לחסימת התנגשויות (Mutual Exclusion) בעת גישה לערימה מסוימת.

15.12 ניהול הזיכרון של תהליך מוגדר באמצעות פונקציות הערימה

כפי שלמדת, התוכניות צריכות להקצות כמות קטנה של זיכרון מערימת הזיכרון שמשמשת את התהליך. התוכניות משתמשות בדרך כלל בפונקציה HeapAlloc להקצאת בלון זיכרון כזה מתוך הערימה. הזיכרון שמוקצה על ידי HeapAlloc אינו בר-העברה (Not Movable). את הפונקציה HeapAlloc כותבים בתוכנית כמו בדוגמה זו:

```
LPVOID HeapAlloc(
  HANDLE hHeap,          // handle to the private heap block
  DWORD dwFlags,         // heap allocation control flags
  DWORD dwBytes          // number of bytes to allocate
);
```

טבלה 15.7: הדגלים עבור הפרמטר **dwFlags**.

דגל	פירוש
HEAP_GENERATE_EXCEPTIONS	מציין שמערכת ההפעלה תיצור חריגה (Exception) כדי לציין כישלון פונקציה, ולא תחזיר NULL, כמו למשל במצב שאין מספיק זיכרון (Out-Of-Memory).
HEAP_NO_SERIALIZE	מציין שהערימה לא תשתמש באפשרות מניעת התנגשות (Mutual Exclusion) כאשר הפונקציה HeapCreate ניגשת לערימה.
HEAP_ZERO_MEMORY	מציין ש-Windows תאתחל את הזיכרון שהוקצה, בערך אפס.

הפרמטר hHeap מגדיר את הערימה שממנה מקצה HeapAlloc את הזיכרון. פרמטר זה הוא ידית המוחזרת על ידי אחד משתי הפונקציות HeapCreate או GetProcessHeap. הפרמטר dwFlags מגדיר מספר דרכים לשליטה בהקצאת זיכרון מהערימה. קביעת אחד משני הדגלים האלה עוקפת את הדגל שצוין בזמן יצירת הערימה עם HeapCreate. באפשרותך להגדיר דגל אחד, או יותר, עבור הפרמטר dwFlags, מאלה המפורטים בטבלה 15.7.

לבסוף, הפרמטר dwBytes מגדיר את מספר הבתים ש-HeapAlloc מקצה. אם הפרמטר hHeap מגדיר ערימה "שאינה צומחת", dwBytes חייב להיות פחות מ- 0x7FFF8. קוראים לפונקציה HeapCreate עם ערך שאינו אפס, כדי ליצור ערימה "שאינה צומחת". אם הפונקציה מצליחה, הערך המוחזר הוא מצביע לבלוק הזיכרון שהוקצה; אם הפונקציה נכשלת והדגל HEAP_GENERATE_EXCEPTIONS לא הוגדר, הערך המוחזר הוא NULL. אם הפונקציה נכשלת והדגל HEAP_GENERATE_EXCEPTIONS לא הוגדר, הפונקציה אולי תיצור חריגה עם ערך של שגיאה, כפי שמפורט בטבלה 15.8.

טבלה 15.8: ערכי השגיאה של הקצאה שגויה בערימה.

ערך	פירוש
STATUS_NO_MEMORY	ההקצאה שרצינו לבצע נכשלה בגלל חיסרון בזיכרון הזמין או פגם בערימה.
STATUS_ACCESS_VIOLATION	ההקצאה שרצינו לבצע נכשלה בגלל פגם בערימה, או פרמטרים שאינם מוגדרים היטב.

שים לב שפגם ערימה יכול לגרום לאחת משתי החריגות; הדבר תלוי בסוג הפגם. אם HeapAlloc מצליחה, היא מקצה לפחות את כמות הזיכרון שדרשה התוכנית הקוראת. אם הכמות הממשית ש-HeapAlloc מקצה יותר גדולה מהכמות שנדרשה על ידי התוכנית הקוראת, התהליך יכול להשתמש בכל הכמות. באפשרותך להשתמש בפונקציה HeapSize, כדי לקבוע את הגודל הממשי של הבלוק שהוקצה.

כדי לשחרר בלוק זיכרון שהוקצה על ידי HeapAlloc, צריך להשתמש בפונקציה HeapFree. זיכרון שהוקצה על ידי HeapAlloc אינו בר-העברה. מכיון שהזיכרון אינו בר-העברה, אפשר שהערימה תהפוך למקוטעת (Fragmented).

<u>שים לב:</u>

אם לא מגדירים את הדגל HEAP_ZERO_MEMORY, Windows אינה מאתחלת לאפס את הזיכרון שהוקצה.

כדי להבין טוב יותר את פעולת הפונקציה HeapAlloc, התבונן בתוכנית **Heap_Strings**, שנמצאת בתקליטור (Chap15).התוכנית יוצרת ערימה, אחר כך משתמשת ב-HeapAlloc כדי להקצות זיכרון מהערימה שנוצרה. היא מתייחסת לזיכרון שהוקצה כמערך דינמי של מחרוזות. כאשר המשתמש בוחר את האפשרות !Allocate מהתפריט, התוכנית מקצה זיכרון מהערימה, כדי לאחסן בו מחרוזת חדשה. היא גם מצהירה על מצביע לזיכרון שהקצאה הקודמת הוסיפה למערך. אם לא נשאר מקום פנוי במערך, התוכנית משתמשת ב-HeapReAlloc כדי להרחיב אותו. כאשר המשתמש בוחר באפשרות !Free מהתפריט, התוכנית משחררת את הזיכרון שהוקצה למחרוזת האחרונה. כאשר התוכנית מגלה שהמשתמש שחרר מספיק זיכרון כדי להשאיר כמות מספקת של מקום שאינו מנוצל, התוכנית מקצה מחדש את הערימה כדי לצמצם את המערך. בנוסף, בכל פעם שהתוכנית מקצה מחדש את הערימה, היא משתמשת בפונקציה HeapCompact כדי לדחוס אותה.

15.13 בדיקת גודל הזיכרון שהוקצה מתוך הערימה

כפי שלמדת, תוכניות Windows מקצות פעמים רבות כמות קטנה של זיכרון לוקלי מערימה פרטית. בסעיפים קודמים יצרנו ערימה והקצאנו זיכרון מתוכה. התוכניות יכולות להשתמש גם בפונקציה HeapReAlloc להקצאה מחדש של שטחי זיכרון מתוך הערימה, וב-HeapFree - כדי לשחרר זיכרון שהוקצה מהערימה. בנוסף, התוכניות צריכות להשתמש תמיד בפונקציה HeapDestroy כדי לפרק ערימות פרטיות שיצרו. אך פעמים רבות, נרצה בזמן הריצה של התוכניות לבדוק את גודל ההקצאה שנעשתה מהערימה. כדי לעשות זאת צריך להשתמש בפונקציה HeapSize, אשר מאפשרת לבדוק את גודל בלוק הזיכרון שהוקצה מהערימה. הפונקציה HeapSize מחזירה את הגודל, בבתים, של בלוק זיכרון שהוקצה מהערימה על ידי הפונקציות HeapAlloc או HeapReAlloc. את הפונקציה HeapSize כותבים כמו בדוגמה שלהלן:

```
DWORD HeapSise(
  HANDLE hHeap,      // handle to the heap
  DWORD dwFlags,     // heap size control flags
  LPCVOID lpMem      // pointer to memory to return size for
);
```

הפרמטר hHeap מגדיר את הערימה שבלוק הזיכרון נמצא בה. אחת משתי הפונקציות HeapCreate או GetProcesHeap מחזירות את הידית הזאת. הפרמטר dwFlags מגדיר מספר דרכים לשליטה בגישה לבלוקים של הזיכרון. כרגע, באפשרותך להגדיר את הדגל HEAP_NO_SERIALIZE ; אך Windows שומרת את שאר ערכי הדגלים לשימוש עתידי. באופן ספציפי הדגל HEAP_NO_SERIALIZE עוקף (Override) את הדגל המתאים שהוגדר בפרמטר flOptions כאשר השתמשת בפונקציה HeapCreate ליצירת הערימה. לבסוף, הפרמטר lpMem מצביע לבלוק הזיכרון שהפונקציה צריכה לחשב את הגודל שלו. זהו מצביע שהפונקציה HeapAlloc או HeapReAlloc מחזירות.

אם הפונקציה HeapSize מצליחה, הערך המוחזר הוא הגודל, בבתים, של בלוק זיכרון שהוקצה ; אם הפונקציה נכשלת - הערך המוחזר הוא 0xFFFFFFFF.

כדי להבין יותר טוב את פעולת הפונקציה HeapSize, התבונן בתוכנית **Heap_Size**, שנמצאת בתקליטור המצורף לספר זה (בתיקיה Chap15). התוכנית יוצרת ערימה ומקצה בלוק זיכרון באורך 20 בתים שמאותחל באפס. אחר כך, התוכנית קוראת לפונקציה HeapSize כדי להציג את גודל הבלוק שהוקצה.

15.14 הקצאת בלוק זיכרון וירטואלי

בתוכניות הפועלות תחת Windows, מקצים בדרך כלל זיכרון על ידי שימוש באחד משלושה סוגי הקצאה : הקצאה מהערימה הגלובלית, הקצאה מהערימה הפרטית, או הקצאה ישירה של זיכרון וירטואלי.

ראית שהשימוש בזיכרון וירטואלי מקנה לתוכניות אפשרויות נוספות ושליטה על תהליך ההקצאה. אך כמו בשאר סוגי ההקצאה, הקצאת הזיכרון נעשית עם פונקציות alloc. הפונקציה VirtualAlloc שונה מפונקציות הקצאה אחרות בזה שהיא **מקצה למלאי** (Reserves) או שהיא **משריינת** (Commits) אזור של דפים במרחב הזיכרון הווירטואלי של התהליך הקורא. Windows מאתחלת באופן אוטומטי באפס את הזיכרון שהוקצה בתוכניות על ידי השימוש בפונקציה VirtualAlloc. את הפונקציה VirtualAlloc כותבים כמו בהגדרה שלהלן :

```
LPVOID VirtualAlloc(
  LPVOID lpAddress   // address of region to reserv or commit
  DWORD dwSize,                   // size of region
  DWORD flAllocationType,         // type of allocation
  DWORD flProtect                 // type of access protection
);
```

הפונקציה VirtualAlloc מקבלת את הפרמטרים שמפורטים בטבלה 15.9.

טבלה 15.9: הפרמטרים של הפונקציה **VirtualAlloc**.

פרמטר	תיאור
lpAddress	מציין את כתובת ההתחלה של האזור שרוצים להקצות. אם התוכנית מקצה את הזיכרון כמלאי, Windows מעגלת כלפי מטה את הכתובת המוגדרת לגבול של 64KB הבאים. אם התוכנית כבר שמרה את הזיכרון שהוקצה כמלאי, והיא קוראת כעת ל-VirtualAlloc כדי לשריין זיכרון, Windows מעגלת כלפי מטה לגבול הדף הבא. כדי לדעת את גודל הדף במחשב המארח, השתמש בפונקציה GetSystemInfo. אם פרמטר זה שווה ל-NULL, המערכת קובעת היכן להקצות את האזור.
dwSize	מציין את גודל האזור, בבתים. אם הפרמטר lpAddress שווה ל-NULL, Windows מעגלת כלפי מעלה את ערך dwSize לגבול הדף הבא; אחרת, הדפים המוקצים יכילו את כל הדפים שמכילים בית אחד או יותר בתחום מ-lpAddress עד (lpAddress + dwSize). כלומר, תחום 2 בתים שנכלל בגבול הדף גורם ל-Windows לכלול שני דפים באזור המוקצה.
flAllocationType	מציין את סוג ההקצאה. באפשרותך להגדיר כל צירוף דגלים המפורטים בטבלה 15.10 .
flProtect	מציין את סוג הגנת הגישה (Access Protection). כאשר משתמשים ב-VirtualAlloc כדי לשריין דפים, באפשרותך להגדיר כל אחד מהדגלים שמפורטים בטבלה 15.11, יחד עם מצייני דגלי ההגנה PAGE_GUARD או PAGE_NO_CACHE כפי שדרוש.

מתוך טבלה 15.9 למדת שמשתמשים בפרמטר flAllocationType כדי לשלוט בהקצאה שמבצעת VirtualAlloc. באפשרותך לציין כל צירוף של הדגלים שמפורטים בטבלה 15.10, כדי לשלוט בהקצאת זיכרון וירטואלית.

טבלה 15.10: הערכים האפשריים של סוגי ההקצאות עבור הפונקציה VirtualAlloc.

דגל	פירוש
MEM_COMMIT	מקצה אחסון פיסי בזיכרון או בקובץ הדפדוף שבדיסק עבור אזור הדפים המוגדר. כלומר, הוא מגן על חלק ממרחב הכתובות של הזיכרון הווירטואלי של התהליך מפני קריאות הקצאה אחרות של אותו תהליך. הניסיון לשריין (Commit) דפים שכבר שוריינו, אינו גורם לכישלון הפונקציה. כלומר, התוכנית יכולה לשריין תחום של דפים שכבר שוריינו, או לשריין דפים מחדש מבלי לדאוג לכישלון.

דגל	פירוש
MEM_RESERVE	שומר כמלאי תחום של מרחב כתובות בזיכרון הווירטואלי של התהליך, מבלי להקצות אמצעי אחסון פיסי כלשהו. כל הקצאה אחרת (כמו של הפונקציה malloc, או של הפונקציה GlobalAlloc) אינה יכולה להשתמש במלאי זה עד שהתוכנית משחררת את התחום. התוכניות יכולות לשריין דפים מהמלאי על ידי קריאות עוקבות לפונקציה VirtualAlloc.
MEM_TOP_DOWN	מקצה זיכרון בכתובת הגבוהה ביותר האפשרית.

בגלל דרך ההקצאה של הדף הווירטואלי, באפשרותך לשלוט בגישה לדפים הווירטואליים ששוריינו על ידי הפונקציה VirtualAlloc. כמו שרואים בטבלה 15.11, באפשרותך להגדיר סוג אחד של הגנה לדף (Page Security), יחד עם המציינים PAGE_GUARD ו-PAGE_NOCACHE. טבלה 15.11 מציגה את דגלי ההגנה שבאפשרותך להשתמש בהם כאשר אתה מקצה דפים וירטואליים.

טבלה 15.11: דגלי ההגנה האפשריים עבור הקצאות דף וירטואליות.

דגל	פירוש
PAGE_READONLY	מאפשר מצב גישה לקריאה בלבד לאזור הדפים ששוריינו. הניסיון לכתוב לדפי קריאה בלבד גורם לשגיאת גישה. אם המערכת מבדילה בין גישה לקריאה בלבד וגישה להפעלה, הניסיון להפעיל קוד באזור החיוב גורם לשגיאת גישה (Access Violation).
PAGE_READWRITE	מאפשר שני מצבי גישה: גישה לקריאה וגישה לכתיבה לדפים באזור הדפים ששוריינו.
PAGE_EXECUTE	מאפשר רק מצב גישה להפעלת דפים באזור המשוריין. הניסיון לקרוא או לכתוב לדפי הפעלה בלבד גורם לשגיאת גישה.
PAGE_EXECUTE_READ	מאפשר שני מצבי גישה: גישה להפעלה וגישה לקריאה לדפים באזור המשוריין. הניסיון לכתוב לדפי הפעלה וקריאה בלבד גורם לשגיאת גישה.
PAGE_EXECUTE_READWRITE	מאפשר את הגישה להפעלה, לקריאה, וכתיבה לדפים באזור המשוריין.

דגל	פירוש
PAGE_GUARD	דפים באזור נעשים **דפים שמורים** (Guard Pages). אם תוכנית מנסה לקרוא מדף שמור או לכתוב בדף כזה, הדף השמור גורם למערכת ההפעלה להדליק את דגל החריגה של מצב שמירה STATUS_GUARD_PAGE ולכבות את מצב הדף השמור. לכן, דפים שמורים משמשים להתראת גישה חד-פעמית. הדגל PAGE_GUARD הוא מציין הגנה לדף. היישום משתמש בו עם אחד משאר דגלי ההגנה של הדף, אך עם הבדל אחד : היישום אינו יכול להשתמש בו עם PAGE_NOACCESS. אחרי קריאה שנכשלה, או פעולת כתיבה הגורמת למערכת ההפעלה לכבות את מצב דף שמור, הגנת הדף שברקע מופעלת. אם מתרחשת חריגה של דף שמור בזמן קבלת שירות מהמערכת, השירות מחזיר בדרך כלל מציין מצב של כישלון (בסעיף 15.15 תמצא הסבר אודות דפים שמורים).
PAGE_NOACCESS	חוסם כל גישה לאזור הדפים שוריינו. הניסיון לקרוא מדף, לכתוב לדף, או להפעיל דף באזור שהגישה אליו נחסמה - גורם לשגיאת גישה, שנקראת שגיאת **הגנה כללית** (GP, General Protection).
PAGE_NOCACHE	מאפשר לא לשנות את הדפים שהוקצו קודם עם הדגל MEM_COMMIT. צריך להגדיר את מאפייני החומרה לזיכרון הפיסי, כמו לדוגמה, "ללא מטמון" (No Cache). מיקרוסופט אינה ממליצה על שימוש כללי בדגל זה. דגל זה יכול לשרת מנהלי התקן (לדוגמה, מיפוי מאגר של מסגרת וידיאו) שאינם משתמשים במטמון. דגל זה הוא מציין הגנה לדף, שתקף רק כאשר משתמשים בו בשילוב עם אחד מהגנות הדף השונותם מ-PAGE_NOACCESS.

אם הפונקציה VirtualAlloc מצליחה, הערך המוחזר הוא כתובת הבסיס של אזור הדפים שהוקצה; אם הפונקציה נכשלת, הערך המוחזר הוא NULL.

VirtualAlloc יכולה לבצע את הפעולות הבאות:

- ☆ לשריין אזור של דפים שקריאה קודמת לפונקציה VirtualAlloc העבירה למלאי.
- ☆ להעביר למלאי אזור של דפים חופשיים.
- ☆ להעביר למלאי ולשריין אזור של דפים חופשיים.

באפשרותך להשתמש ב-VirtualAlloc כדי להעביר למלאי בלוק של דפים, ואחר כך לקרוא שוב ל-VirtualAlloc כדי לשריין דפים בודדים מבלוק המלאי. קיום בלוק של דפים מאפשר לתהליך לשמור על תחום הכתובת הווירטואלית שלו, מבלי לנצל אחסון פיסי עד שנזקקים לו.

כל דף במרחב הזיכרון הווירטואלי של התהליך נמצא באחד משלושה מצבים שמפורטים בטבלה 15.12.

טבלה 15.12: המצבים האפשריים של זיכרון וירטואלי.

מצב	פירוש
Free חופשי	התהליך לא שריין או לא שמר למלאי דף זה, ולכן הדף אינו נגיש לתהליך. VirtualAlloc יכולה לשמור במלאי, או בו-זמנית לשמור במלאי וגם לשריין דף חופשי.
Reserved שמור כמלאי	פונקציות הקצאה אחרות אינן יכולות להשתמש בתחום הכתובות, אבל היישום אינו יכול לגשת לתחום ו-Windows אינה משייכת אמצעי אחסון פיסי עם הדף. VirtualAlloc יכולה לשריין דף שבמלאי, אבל אינה יכולה להעביר אותו למלאי פעם נוספת. הפונקציה VirtualAlloc יכולה לשחרר דף הנמצא במלאי, ולהפוך אותו לדף חופשי.
Committed משוריין	Windows מקצה זיכרון עבור דף, וקוד מוגן שולט בגישה. המערכת מאתחלת וטוענת כל דף משוריין לזיכרון הפיסי רק בניסיון הראשון לקרוא או לכתוב לדף זה. כאשר התהליך מסתיים, המערכת משחררת את האחסנה של הדפים המשוריינים. VirtualAlloc יכולה לשריין דף שכבר שוריין. כלומר, אפשר לשריין תחום דפים, ללא קשר אם כבר שוריינו, והפונקציה אינה נכשלת. VirtualFree יכולה לבטל שריון של דף משוריין, לשחרר אמצעי האחסנה של הדפים, או שהיא יכולה בו-זמנית גם לבטל שריון וגם לשחרר דף ששוריין.

אם הפרמטר lpAddress הוא NULL, הפונקציה משתמשת בפרמטרים lpAddress ו-dwSize כדי לחשב את אזור הדפים ש-VirtualAlloc מקצה. המצב הנוכחי של כל תחום הדפים חייב להיות מתאים לסוג ההקצאה שמוגדר על ידי הפרמטר flAllocationType; אחרת, הפונקציה נכשלת ו-VirtualAlloc אינה מקצה את הדפים. דרישת התאימות אינה מונעת שריון דפים שכבר שוריינו (ראה טבלה 15.12).

כדי להבין טוב יותר את פעולת הפונקציה VirtualAlloc, התבונן בתוכנית **Virtual_Allocate**, שנמצאת בתקליטור המצורף לספר זה (בתיקיה Chap15). תוכנית זו מעבירה למלאי מגהבית אחד (1MB) של זיכרון וירטואלי, כאשר היא שולחת את ההודעה WM_CREATE. כאשר המשתמש בוחר באפשרות התפריט !Test, התוכנית משריינת ומשתמשת ב- 70KB של זיכרון וירטואלי. ראשית, התוכנית מציבה ערכים בכל בלוק של KB בזיכרון שהוקצה. שנית, התוכנית משנה לקריאה בלבד, את אפשרות הגישה לכל בלוק הזיכרון ששוריין. שלישית, התוכנית ניגשת לערך בזיכרון ומציגה אותו בתיבת הודעה. לבסוף, התוכנית מנסה להציב ערך בזיכרון, והדבר גורם לשגיאת הגנה. התוכנית **Virtual_Allocate** משתמשת בבלוק הקוד try-catch כדי לגלות את שגיאת ההגנה.

15.15 דפים שמורים (Guard Pages)

למדת בסעיף 15.14 שיישום קובע את דגל מציין ההגנה PAGE_GUARD עבור דף זיכרון, כדי להגדיר **דף שמור** (Guard Page). באפשרותך להגדיר דגל זה יחד עם דגלי הגנה אחרים לדפים, באמצעות הפונקציות VirtualAlloc, VirtualProtect ו-VirtualProtectEx. תוכל גם להשתמש בדגל PAGE_GUARD עם כל דגל הגנה אחר לדפים, מלבד הדגל PAGE_NOACCESS.

אם תוכנית מנסה לגשת לכתובת שנמצאת **בדף שמור** (Guard Page), מערכת ההפעלה יוצרת מצב חריג STATUS_GUARD_PAGE (0x80000001). מערכת ההפעלה גם מוחקת את דגל PAGE_GUARD, ומסירה את המצב השמור מדפי הזיכרון. המערכת אינה מפסיקה את הניסיון הבא לגשת לדפי הזיכרון עם המצב החריג STATUS_GUARD_PAGE.

אם מתרחש מצב חריגה של דף שמור בזמן קבלת שירות מהמערכת, השירות מופסק ובדרך כלל מחזיר ערך כלשהו שמציין מצב כישלון. מכיון שהמערכת מסירה את המצב השמור מדפי הזיכרון, הקריאה הבאה לאותו שירות של המערכת לא תיכשל בגלל מצב חריג STATUS_GUARD_PAGE (אלא אם כן, מישהו אחר החזיר את המצב השמור).

לכן, דף שמור מספק התראה חד-פעמית בעת גישה לדפי זיכרון. הדבר עשוי להיות שימושי ליישומים שחייבים לבקר גידול של מבני נתונים דינמיים גדולים. לדוגמה, חלק ממערכות ההפעלה משתמש **בדפים שמורים** (Guard Pages) כדי ליישם בדיקת מחסנית אוטומטית.

בתקליטור המצורף לספר זה תמצא את התוכנית **Guard_Page** (בתיקיה Chap15), שמדגימה את ההתנהגות החד-פעמית של הגנת דף שמור, וכיצד הוא יכול לגרום לכישלון שירות של המערכת (התוכנית יוצרת קלט לחלון DOS). כשמהדרים ומפעילים את התוכנית **Guard_Page**, היא מציגה במסך את הפלט הבא:

```
committed 512 bytes at address 003F0000
Cannot lock at 003F0000, error = 0x80000001
2nd Lock Achieved at 003F0000
C:\>
```

שים לב שהניסיון הראשון לנעול את בלוק הזיכרון גורם ליצירת מצב חריג STATUS_GUARD_PAGE. הניסיון השני מצליח, מכיון שהניסיון הראשון הסיר את ההגנה של דף שמור מבלוק הזיכרון.

זיכרון וירטואלי מקנה אמצעי נוסף ויעיל להקצאת בלוקים גדולים של זיכרון. עם זאת, הצגת היתרונות של זיכרון וירטואלי לשימוש בתכנות קשה ומורכבת בדרך כלל. אחת הדרכים הקלות להצגת יתרונות הזיכרון הווירטואלי היא לחשוב על מערך גדול של מבנה מורכב, אולי מבנה הדומה לגיליון אלקטרוני. נניח שצריך ליצור מערך דו-ממדי, ההגדרה יכולה להראות דומה לזו שלהלן:

```
Cell LARGEARRAY[200][256];
```

אם גודל המבנה Cell הוא 128 בתים, דרושים לאחסנה 65,533,600 בתים של זיכרון פיסי (לפי החישוב: 200 x 256 x 128). ברור, שזו כמות משמעותית של זיכרון פיסי שצריך להקצות לגיליון האלקטרוני - במיוחד כאשר רוב גליונות העבודה לא ישתמשו בכמות תאים שמתקרבת להקצאת תאים רבה כזאת.

כתחליף, אפשר להשתמש ברשימה מקושרת כדי ליצור את סוג המבנה של הגיליון האלקטרוני. על פי התפיסה של רשימה מקושרת, צריך ליצור מבנה Cell לתאים שבגיליון האלקטרוני אשר מכילים נתונים, ולא לתאים הריקים. עם זאת, השימוש ברשימה מקושרת מקשה מאוד על השגת תוכן התאים. לדוגמה, אם התוכנית דורשת את התוכן התא בשורה 5 שעמודה 10, חייבים לעבור דרך הרשימה המקושרת כדי למצוא את התא, ומכאן ההאטה בהפעלה.

זיכרון וירטואלי מציע פשרה בין ההגדרה של המטריצה הדו-מימדית לבין מימוש מורכב של רשימות מקושרות גדולות. עם זיכרון וירטואלי מקבלים את הגישה המהירה ואת פשטות הגישה הנובעת מטכניקת המטריצה, וגם את יכולת האחסון המעדיפה של רשימה מקושרת. כדי להעריך טוב יותר את היתרונות של טכניקת הזיכרון הווירטואלי, התוכנית צריכה לבצע את הדברים שלהלן:

1. לשמור אזור מלאי שיהיה גדול במידה מספקת כדי להכיל את כל המטריצה של המבנים מסוג Cell. כפי שלמדת, שמירת אזור כמלאי אינה צורכת זיכרון פיסי.
2. להקצות כתובת זיכרון באזור המלאי שבו המבנה Cell צריך להיות כאשר המשתמש מכניס נתונים לתוך תא כלשהו.
3. לשריין עבור המבנה Cell אמצעי אחסנה פיסי מספיקים המיועדים לכתובת הזיכרון שהוקצתה בצעד 2.
4. להציב את האיברים של מבנה Cell החדש.

אחרי ביצוע המיפוי הפיסי למקום המתאים, התוכנית יכולה לגשת אל אמצעי האחסון מבלי ליצור שגיאת גישה (Access Violation). ברור, שטכניקת הזיכרון הווירטואלי היא שיפור משמעותי לעומת שאר הטכניקות, מפני שהתוכנית משריינת אמצעי אחסון פיסי רק כאשר המשתמש מכניס נתונים לתאים של הגיליון האלקטרוני. מכיון שרוב התאים בגיליון האלקטרוני ריקים, התוכנית אינה משתמשת במרבית אזור המלאי. באפשרותך להקצות זיכרון וירטואלי כדי שהתוכנית תוכל לגשת מהר יותר למספר גדול של איברים, בלי הקשיים והוויתורים שנעשים במודלים קודמים של זיכרון.

15.17 שחרור זיכרון וירטואלי

למדת שבאפשרות התוכנית להשתמש בפונקציה VirtualAlloc כדי להעביר למלאי או לשריין דפי זיכרון וירטואלי. לאחר העברה למלאי או לאחר השריון של דפי זיכרון וירטואלי, התוכנית יכולה להשתמש בפונקציה VirtualFree כדי לשחרר או לבטל חיוב של דפים אלה. הפונקציה VirtualFree משחררת אזור של דפים או מבטלת את השריון שלהם (או שניהם) במרחב הכתובות הווירטואליות של התהליך הקורא. משתמשים בפונקציה VirtualFree כמו שרואים בהגדרה שלהלן:

```
BOOl VirtualFree(
  LPVOID lpAddress       // address of region of committed pages
  DWORD dwSize,          // size of region
  DWORD dwFreeType,      // type of free operation
);
```

הפרמטר lpAddress מצביע לכתובת הבסיס של אזור הדפים שהפונקציה VirtualFree אמורה לשחרר. אם הפרמטר dwFreeType מכיל את הדגל MEM_RELEASE, הפרמטר lpAddress חייב להיות כתובת הבסיס שהפונקציה VirtualAlloc החזירה כאשר העבירה למלאי את אזור הדפים. הפרמטר dwSize מגדיר את הגודל, בבתים, של האזור ש-VirtualFree אמורה לשחרר. אם הפרמטר dwFreeType מכיל את הדגל MEM_RELEASE, הפרמטר dwSize חייב להיות אפס; אחרת, אזור הדפים שעומדים לטפל בהם יכיל את כל הדפים שמכילים בית אחד, או יותר, בתחום שנמצא בין ערך הפרמטר lpAddress ועד lpAddress+dwSize. כלומר, תחום של 2 בתים שנמצא בגבול הדף גורם ל-VirtualFree לשחרר את שני הדפים. dwFreeType מגדיר את סוג פעולת השחרור. כאשר קוראים לפונקציה VirtualFree, משתמשים בדגל אחד, אך לא בשני הדגלים שמפורטים בטבלה 15.13.

טבלה 15.13: הדגלים עבור הפרמטר dwFreeType.

דגל	פירוש
MEM_DECOMMIT	מבטל את שריון אזור מוגדר של דפים משוריינים. ניסיון לבטל שריון דפים שאינם משוריינים אינו גורם לכישלון הפונקציה. כלומר, התוכניות יכולה לבטל שריון תחום של דפים ששוריינו, או של דפים שלא שוריינו קודם, מבלי שיהיה צורך לדאוג לכישלון.
MEM_RELEASE	משחרר את האזור המוגדר של דפים שבמלאי. אם התוכנית מגדירה דגל זה, הפרמטר dwSize חייב להיות אפס, או שהפונקציה תיכשל.

התוכניות יכולה להשתמש בפונקציה VirtualFree כדי לבצע אחת משלוש הפעולות האלו:

- ✰ לבטל שריון אזור של דפים משוריינים, או של דפים שאינם משוריינים.
- ✰ לשחרר אזור של דפים במלאי.
- ✰ לבטל שריון ולשחרר אזור של דפים משוריינים, או דפים שאינם משוריינים.

כדי לשחרר אזור של דפים, כל תחום הדפים חייב להיות באותו מצב (כולם במלאי, או כולם משוריינים) והפונקציה VirtualFree חייבת לשחרר את כל אזור המלאי המקורי באותו זמן. אם השתמשת קודם ב-VirtualAlloc כדי לשריין רק חלק מהדפים שבאזור המלאי המקורי, עליך לקרוא ל-VirtualFree כדי לבטל את שריון הדפים המשוריינים, ואחר כך לקרוא ל-VirtualFree שוב כדי לשחרר את כל הבלוק.

הדפים ששוחררו על ידי שימוש ב-VirtualFree הם דפים חופשיים, שתקפים לפעולות של הקצאות עוקבות. ניסיון לכתוב לדף חופשי או לקרוא מדף חופשי, יגרום להודעת חריגה מסוג שגיאת גישה (Access Violation). VirtualFree יכולה לבטל שריון של דף שאינו משוריין; כלומר, הפונקציה VirtualFree יכולה לבטל שריון של תחום דפים כלשהו הכולל דפים משוריינים ושאינם משוריינים, מבלי שיהיה צורך לדאוג לכישלון. ביטול השריון של דף משחרר את האחסון הפיסי שלו שנמצא בזיכרון או בקובץ הדפדוף בדיסק. אם התוכנית מבטלת שריון דף אבל אינה משחררת אותו, מצב הדף משתנה למלאי וקריאה עוקבת ל-VirtualAlloc יכולה לגרום לשריון חוזר שלו. הניסיון לקרוא מדף או לכתוב לדף שבמלאי גורם להודעת חריגה מסוג שגיאת גישה.

המצב הנוכחי של כל תחום הדפים חייב להיות מתאים לסוג פעולת השחרור שמוגדרת על ידי הפרמטר dwFreeType. אחרת, הפונקציה VirtualFree נכשלת ולא משחררת, או לא מבטלת שריון של דף כלשהו.

15.18 ניהול דפי זיכרון וירטואלי

דפי זיכרון וירטואלי מיועדים להרחיב את גישת התוכניות לזיכרון וגם לעזור לתוכניות בהקצאת בלוקים גדולים של זיכרון במרחב הזיכרון הווירטואלי. פעמים רבות, נדרש מידע על תחום הדפים שבתחום מרחב כתובות הזיכרון הווירטואלי של התהליך. הפונקציה VirtualQuery מספקת מידע על תחום הדפים שבתחום מרחב הזיכרון הווירטואלי של התהליך הקורא, כמו שרואים להלן:

```
DWORD VirtualQuery(
  LPCVOID lpAddress                       // address of region
  PMEMORY_BASIC_INFORNATION lpBuffer,     // address of
                                          // information buffer
  DWORD dwLength,                         // size of buffer
);
```

הפונקציה VirtualQuery מקבלת שלושה פרמטרים, כמפורט בטבלה 15.14.

טבלה 15.14: הפרמטרים של הפונקציה **VirtualQuery**.

פרמטר	תיאור
lpAddress	מצביע לכתובת הבסיס של אזור הדפים ש-VirtualQuery בודקת. Windows מעגלת ערך זה כלפי מטה, לגבול הדף הבא. כדי לדעת מהו גודל הדף במחשב המארח, השתמש בפונקציה GetSystemInfo.
lpBuffer	מצביע למבנה מסוג MRMORY_BASIC_INFORMATION ש-VirtualQuery מחזירה בו את המידע על תחום הדפים המוגדר.
dwLength	מגדיר את גודל המאגר, בבתים, שמצביע עליו הפרמטר lpBuffer.

VirtualQuery מספקת מידע על אזור של דפים עוקבים מכתובת התחלתית מוגדרת, בעלת המאפיינים הבאים:

- המצב של כל הדפים אותו דבר עם הדגלים MEM_COMMIT, MEM_RESERVE, MEM_FREE, MEM_PRIVATE, MEM_MAPPED, או MEM_IMAGE.
- אם הדף ההתחלתי אינו חופשי, כל הדפים שבאזור יהיו חלק של ההקצאה הראשונית של דפים שהקריאה של הפונקציה VirtualAlloc העבירה למלאי.
- הגישה של כל הדפים אותו דבר עם הדגלים PAGE_READONLY, PAGE_READWRITE, PAGE_NOACCESS, PAGE_WRITECOPY, PAGE_EXECUTE, PAGE_EXECUTEREAD, PAGE_GUARD, PAGE_EXECUTE_READWRITE, PAGE_EXECUTE_WRITECOPY, או PAGE_NOCACH.

הפונקציה VirtualQuery קובעת את מאפייני הדף הראשון באזור. אחר כך היא סורקת דפים עוקבים עד שלסיום הסריקה של כל תחום הדפים, או עד שהיא מוצאת דף עם קבוצת מאפיינים שונה. VirtualQuery מחזירה את המאפיינים ואת גודל אזור הדפים בעל אותם המאפיינים. לדוגמה, אם יש אזור חופשי בגודל 40MB והתוכנית קוראת ל-VirtualQuery עם דף באזור שאחרי 10MB, הפונקציה מציגה סטטוס MEM_FREE וגודל של 30MB.

הפונקציה VirtualQuery מדווחת על אזור דפים בזיכרון של התהליך הקורא, והפונקציה VirtualQueryEx מדווחת על אזור דפים בזיכרון של התהליך המוגדר. בתקליטור שמצורף לספר זה נמצאת התוכנית **Virtual_Query** (בתיקיה Books\59285), שבתחילה היא מקצה בלוק זיכרון וירטואלי בגודל 70KB. אחר כך היא קוראת לפונקציה VirtualQuery, שמחזירה את גודלו הממשי של האזור שתפוס על ידי הזיכרון. שים לב, גודל האזור מתחלק ב- 4096 בתים (4KB), שזה גודל הדף של מחשבי x86.

פרק 16

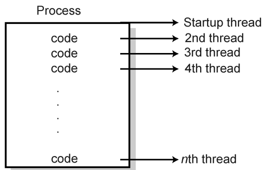
תהליכים ומטלות

16.1 הבנה יותר טובה של תהליכים

אחת התכונות הבולטות ביותר של Windows היא התמיכה שלה ב**ריבוי משימות** (Multitasking), שמשמעותה - הפעלת תהליכים אחדים בזיכרון בו-זמנית. בשלושים הסעיפים הבאים, תלמד יותר על ניהול **תהליכים** (Processes) **ומטלות** (Threads), אחת המיומנות החשובות ביותר למתכנתי Windows.

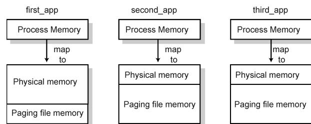
תהליך (Process) הוא אובייקט הבעלים של כל משאבי היישום. תהליך של Windows, בתורו, יכול ליצור מטלה אחת או יותר. **מטלה** (Thread) היא נתיב ביצוע בלתי תלוי בתוך תהליך, אשר מנצל חלק ממרחב הכתובות שלו, קוד ונתונים גלובליים. לכל מטלה יש קבוצת **אוגרים** (Regeisters) נפרדת, וגם מנגנון קלט נפרד הכולל תור הודעות פרטי. Windows 9x ו-Windows NT מקצות קטעי זמן מעבד (CPU) לכל מטלה בנפרד, **מטלה-אחרי-מטלה** (Thread-By-Thread) וכך הן מנהלות את מנגנון ריבוי המשימות: מפעילות כל משימה על פי העדיפות שמוקצית לכל מטלה. אפשר לתאר זאת כקבוצת מטלות שמסודרות זו אחר זו, ועל פי סדר עדיפות שנקבע, המטלה המבקשת שירות מוצגת לפני כל חברותיה ופועלת, עד אשר מטלה אחרת עדיפה יותר תזכה לקבל את שירותי המערכת (Preemptive Multitasking).

בנוסף, תהליך כולל גם הקצאות זיכרון גלובליות, דפים וירטואליים ועוד. תרשים 16.1 מציג מודל לוגי של הקשרים בין תהליכים ומטלות.



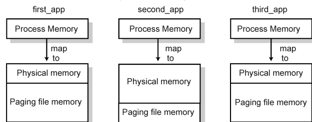
תרשים 16.1: הקשר בין תהליכים ומטלות.

למדת ש-Windows מקצה זיכרון וירטואלי לתהליכים ככל שנדרש להם. לכן, כאשר לוקחים בחשבון את מודל הזיכרון למחשב שמריץ תהליכים רבים בו-זמנית, חשוב לזהות איזה תהליך הוא התהליך **הפעיל** (Active). הקביעה מיהו התהליך הפעיל כרגע חשובה, מכיון ש-Windows מצמידה באופן אוטומטי עדיפות גבוהה יותר לרוב הדרישות לזמן מעבד שיוצאות מטעם התהליך הפעיל. בסעיף 16.25 תמצא דיון מפורט בניהול עדיפויות המטלות על ידי מערכת ההפעלה. בנוסף, Windows מקצה בדרך כלל זיכרון פיסי נוסף, כדי להאיץ את ביצוע התהליך הפעיל כרגע. תרשים 16.2 מציג דוגמה למודל זיכרון עבור שלושה יישומים הפועלים בו-זמנית. היישום הפעיל הוא First_App.



תרשים 16.2: דוגמה למודל זיכרון לשלושה יישומים שפועלים בו-זמנית.

עם זאת, כאשר נוצר מאורע על ידי המשתמש או על ידי Windows, אשר גורם לכך שהתוכנית השנייה Second_App תהיה פעילה, Windows מקצה זיכרון פיסי נוסף עבור התוכנית הזו ונותנת לה מרחב זיכרון רב יותר לפעולה, כמו שמוצג בתרשים 16.3.



תרשים 16.3: דוגמה אחרת למודל זיכרון לשלושה יישומים שפועלים בו-זמנית.

בחמשת הסעיפים הבאים תלמד את הדרכים ליצור ולנהל תהליכים ומטלות. צריך להבין תחילה את יסודות הבניה להפעלה של תהליכים (שבעיקרון אינם אלא מכולות, Containers, של מטלות) לפני שנעבור לנושא המטלות.

16.2 יצירת תהליך

למדת שבדרך כלל התוכניות מופעלות כתהליך אחד, אשר יכול להכיל מטלה אחת או יותר. הפונקציה CreateProcess יוצרת תהליך חדש ואת **המטלה הראשית** שלו (הידועה גם בשם **תהליך-בן**, Child Process). התהליך החדש מפעיל את קובץ ההפעלה שהוגדר. CreateProcess מאפשרת ל**תהליך האב**, Parent Process (כלומר, התהליך שקורא לפונקציה CreateProcess) להגדיר את סביבת העבודה של התהליך החדש, ובכלל זה את תיקיית ההפעלה שלו, ברירת המחדל לתצוגה על המסך, משתני הסביבה והעדיפות. Windows מוסרת את שורת הפקודה ואת תכולתה לתהליך הבן. את הפונקציה CreateProcess כותבים כמו בהגדרה זו:

```
BOOL CreateProcess(
  LPCTSTR lpApplicationName       // name of executable module
  LPTSTR lpCommandLine,           // command line string
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
                                  // process security attributes
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
                                  // thread security attributes
  BOOL bInheritHandles,           // handle inheritance flag
  DWORD dwCreationFlags,          // creation flags
  LPVOID lpEnvironment,           // pointer to new environment
                                  // block
```

```
  LPCTSTR lpCurrentDirectory,   // pointer to current
                                // directory name
  LPSTARTUPINFO lpStartupInfo,  // pointer to STARTUPINFO
  LPPROCESS_INFORMATION lpProcessInformation
                                 // pointer to PROCESS_INFORMATION
);
```

כמו שאפשר לראות, הפונקציה CreateProcess מקבלת פרמטרים רבים. טבלה 16.1 מסבירה את הפרמטרים האלה.

טבלה 16.1: הפרמטרים של הפונקציה CreateProcess.

פרמטר	תיאור
lpApplicationName	מצביע למחרוזת המסתיימת ב-NULL, שמגדירה את המודול שצריך להפעיל. אפשר לשלב במחרוזת זו את הנתיב השלם ואת שם הקובץ של המודול שצריך להפעיל, או שהמחרוזת יכולה להגדיר שם חלקי. כאשר המחרוזת מגדירה שם חלקי בלבד, הפונקציה משתמשת בכונן ובתיקיה הנוכחיים כדי להשלים את ההגדרות. הפרמטר lpApplicationName יכול לקבל את הערך NULL, אך במקרה זה שם המודול חייב להיות הפריט הראשון במחרוזת lpCommandLine, המופרד בתו בקרה (רווח, או תו אחר בלתי נראה, Whitespace). המודול המוגדר יכול להיות יישום מבוסס Win32 או סוג מודול אחר (לדוגמה, MS-DOS או OS/2), אם במחשב המקומי פועלת מערכת ההפעלה המתאימה.
✍ **הערה:**	תחת Windows NT, אם המודול שצריך להפעיל הוא יישום 16 הסיביות, lpApplicationName צריך להיות NULL, והמחרוזת ש-lpCommandLine מצביע עליה צריכה להגדיר את המודול שצריך להפעיל.
lpCommandLine	מצביע למחרוזת המסתיימת ב-NULL שמגדירה את שורת הפקודה שצריך להפעיל. הפרמטר lpCommandLine יכול להיות NULL, אך במקרה כזה הפונקציה משתמשת במחרוזת שהפרמטר lpApplicationName מצביע עליה כשורת הפקודה. אם שני הפרמטרים lpApplicationName ו-lpCommandLine אינם NULL, הפרמטר lpApplicationName מגדיר את המודול שצריך להפעיל, ו-lpCommandLine מצביע לשורת הפקודה. התהליך החדש יכול להשתמש בפונקציה GetCommandLine כדי לקבל את כל שורת הפקודה. תהליכי הביצוע של תהליכים בשפת C (C Run-Time Processes) יכולים להשתמש גם בארגומנטים argc ו-argv.

תיאור	פרמטר
אם lpApplicationName הוא NULL, הרכיב הראשון של שורת הפקודה שמופרד בתו בקרה (רווח או תו בלתי נראה אחר) מגדיר את שם המודול. אם שם הקובץ אינו מכיל **סיומת** (Extension), Windows מניחה שהסיומת היא EXE. אם שם הקובץ מסתיים בנקודה (.) בלי סיומת, או ששם הקובץ מכיל את פירוט הנתיב, Windows אינה מצרפת EXE. אם שם הקובץ אינו מכיל נתיב תיקיה, Windows מחפשת את הקובץ שצריך להפעיל לפי הסדר הבא: 1. התיקיה שממנה נטען היישום. 2. התיקיה הנוכחית של תהליך האב. 3. Windows 9x: תיקיית Windows של המערכת. השתמש בפונקציה GetSystemDirectory לקבלת הנתיב של תיקיה זו. 4. Windows NT: תיקיית 32 הסיביות של מערכת Windows (32-Bit Windows System Directory). השתמש בפונקציה GetSystemDirectory כדי לקבל את נתיב התיקיה. שם התיקיה הוא בדרך כלל SYSTEM32. 5. Windows NT: תיקיית 16 הסיביות של מערכת Windows (16-bit Windows System Directory). אין פונקציה של Win32 שמקבלת את נתיב התיקיה הזו, אך הפונקציה מחפשת בה. שם התיקיה הוא בדרך כלל SYSTEM. 6. התיקיה Windows: כדי לקבל את נתיב התיקיה הזו השתמש בפונקציה GetWindowsDirectory. שם תיקיה זו בדרך כלל הוא Windows. 7. התיקיות שמפורטות במשתנה הסביבה PATH של Windows. כאשר התהליך שאתה רוצה ליצור באמצעות הפונקציה CreateProcess הוא יישום מבוסס MS_DOS או מבוסס Windows, הפרמטר lpCommandLine צריך להיות שורת הפקודה מלאה, שהרכיב הראשון שלה זה הוא שם היישום. מכיון שהפונקציה CreateProcess פועלת גם ביישומים מבוססי Win32, יש לקבוע את הפרמטר lpCommandLine כשורת פקודה מלאה גם עבור תוכניות Win32.	LpCommandLine
מצביע למבנה SECURITY_ATTRIBUTES, שקובע אם תהליך הבן יכול לרשת את הידית המוחזרת. אם lpProcessAttributes הוא NULL, תהליך הבן אינו יכול לרשת את הידית.	lpProcessAttributes

פרמטר	תיאור
✍ הערה:	במערכת Windows NT, האיבר lpSecurityDescriptor של המבנה מגדיר מתאר אבטחה (Security Descriptor) עבור התהליך החדש. אם lpProcessAttributes הוא NULL, התהליך מקבל את ברירת המחדל של מתאר האבטחה. תחת Windows 9x, הפונקציה CreateProcess מתעלמת מאיבר lpSecurityDescriptor של המבנה.
lpThreadAttributes	מצביע למבנה SECURITY_ATTRIBUTES שקובע אם תהליך הבן יכול לרשת את הידית המוחזרת. אם lpThreadAttributes הוא NULL, תהליך הבן אינו יכול לרשת את הידית.
✍ הערה:	במערכת Windows NT, האיבר lpSecurityDescriptor של המבנה מגדיר מתאר אבטחה (Security Descriptor) עבור המטלה הראשית. אם lpThreadAttributes הוא NULL, המטלה מקבלת את ברירת המחדל של מתאר האבטחה. במערכת Windows 9x, הפונקציה CreateProcess מתעלמת מאיבר lpSecurityDescriptor של המבנה.
bInheritHandles	מציין אם התהליך החדש יורש ידיות מהתהליך הקורא. אם כן, כלומר True, התהליך החדש יורש כל ידית פתוחה שאפשר לרשת מהתהליך הקורא. לידיות שעברו בהורשה יש את אותו הערך והגישה כמו לידיות המקוריות.
dwCreationFlags	מגדיר דגלים נוספים ששולטים ב**מחלקת העדיפות** (Priority Class) וביצירת התהליך. אפשר להגדיר את דגלי היצירה המוצגים בטבלה 16.2 בכל צירוף, מלבד אלה שמצוינים בהערות המתאימות. הפרמטר dwCreationFlags קובע גם כן את מחלקת העדיפות החדשה של התהליך, ש-Windows משתמשת בה כדי לקבוע את סדר העדיפויות של מטלות התהליך. אם Windows אינה מגדירה דגל של מחלקת עדיפות כלשהי מאלה שמוצגים בטבלה 16.2, ברירת המחדל שלה היא NORMAL_PRIORITY_CLASS. עם זאת, כאשר מחלקת העדיפות של התהליך היוצר היא IDLE_PRIORITY_CLASS, עדיפות המחלקה של תהליך הבן גם היא IDLE_PRIORITY_CLASS. אפשר להגדיר כל דגל מדגלי העדיפות שמפורטים בטבלה 16.3.

פרמטר	תיאור
lpEnvironment	מצביע לבלוק סביבה של התהליך החדש. אם פרמטר זה הוא NULL, התהליך החדש משתמש בסביבה המוגדרת של התהליך הקורא. בלוק סביבה מורכב מבלוק שמסתיים ב-NULL ומכיל מחרוזות המסתיימות ב-NULL. התבנית של כל מחרוזת הוא name=value. מכיון שהמחרוזת משתמשת בסימן השוויון כאופרטור, אסור להשתמש בסימן זה בשם של משתנה סביבה. אם יישום משתמש בבלוק סביבה, במקום להציב NULL בפרמטר זה, Windows אינה מעבירה לתהליך החדש את המידע של התיקיה הנוכחית אודות כונני המערכת. בלוק סביבה יכול להכיל תווי Unicode או תווי ANSI. אם בלוק הסביבה שמצביע עליו lpEnvironment מכיל תווי Unicode, Windows קובעת את הדגל CREATE_UNICODE_ENVIRONMENT בשדה dwCreationFlags. אם הבלוק מכיל תווי ANSI, דגל זה נמחק. שים לב לכך ששני בתים של אפסים (Two Zero Bytes) מסיימים בלוק סביבה של ANSI: אחד הבתים לסיום המחרוזת האחרונה, ועוד בית אפסים כדי לסיים את הבלוק. לסגירת בלוק Unicode דרושים ארבעה בתים של אפסים: שני בתים עבור המחרוזת האחרונה, ועוד שני בתים של אפסים כדי לסיים את הבלוק.
lpCurrentDirectory	מצביע למחרוזת המסתיימת ב-NULL שמגדירה את הכונן והתיקיה הנוכחיים של תהליך הבן. המחרוזת חייבת להכיל פירוט שלם של הנתיב ושל שם הקובץ, וגם את אות הכונן (לדוגמה, "a:"). אם פרמטר זה הוא NULL, Windows יוצרת את התהליך החדש בכונן ובתיקיה של התהליך הקורא. Windows תומכת באפשרות זו בעיקר עבור מעטפות מערכות הפעלה שחייבות להתחיל בהפעלת יישום ולהגדיר את הכונן ההתחלתי שלו ואת תיקיית ההפעלה.
lpStartupInfo	מצביע למבנה STARTUPINFO המגדיר כיצד החלון הראשי של התהליך החדש צריך להופיע.
lpProcessInformation	מצביע למבנה PROCESS_INFORMATION שמקבל את המידע המזהה עבור התהליך החדש.

כמו שראית בטבלה 16.1, הפרמטר dwFlags מקבל דגל יצירה אחד או יותר ודגל מחלקת עדיפות אחת. טבלה 16.2 מפרטת את דגלי היצירה האפשריים.

טבלה 16.2: דגלי היצירה האפשריים של הפרמטר **dwFlags**.

דגל	פירוש
CREATE_DEFAULT_ERROR_MODE	התהליך החדש אינו יורש את **מצב השגיאה** (Error Mode) של התהליך הקורא. במקום זאת, CreateProcess נותנת לתהליך החדש את מצב השגיאה של ברירת המחדל הנוכחית. היישום קורא ל-SetErrorMode כדי לקבוע את מצב השגיאה של ברירת המחדל הנוכחית. דגל זה שימושי במיוחד עבור **יישומים של מעטפות מרובות מטלות** (Multi-Thread Shell Applications) שאסור להתיר בהן שגיאות חמורות. התנהגות ברירת המחדל של CreateProcess היא שעל התהליך החדש לרשת את מצב השגיאה של התהליך הקורא. שימוש בדגל זה משנה את התנהגות ברירת המחדל הזו.
CREATE_NEW_CONSOLE	התהליך החדש יורש **קונסול** חדש (**console** במינוח החלונאי הוא הממשק של תוכניות מבוססות-טקסט, ובקיצור - מסוף פשוט לא גרפי), ולא יורש את הקונסול של תהליך האב. אי אפשר להשתמש בדגל זה יחד עם הדגל DETACHED_PROCESS.
CREATE_NEW_PROCESS_GROUP	התהליך החדש הוא תהליך השורש של קבוצת תהליכים חדשה. קבוצת התהליכים כוללת את כל התהליכים שהם צאצאים (בנים) של תהליך השורש החדש הזה. המזהה התהליך של קבוצת התהליך החדשה, זהה למזהה התהליך ש-Windows מחזירה בפרמטר lpProcessInformation.
CREATE_SEPARATE_WOW_VDM	ב-Windows NT בלבד: דגל זה תקף רק כאשר מתחילים יישום Windows מבוסס 16 סיביות. אם קובעים דגל זה, Windows מפעילה את התהליך החדש ב**מכונת DOS וירטואלית** פרטית - **V**irtual **D**OS **M**achine) VDM). לפי ברירת המחדל, Windows מריצה את כל יישומי Windows מבוססי 16 סיביות (16-bit Windows-Based Applications) כמטלות ב-VDM יחידה משותפת.

דגל	פירוש
CREATE_SEPARATE_WOW_VDM	היתרון בהפעלת תהליך מכונות DOS וירטואלית נפרדות הוא בכך ש**נפילה** (Crash) **מחסלת** (Kill, פעולת סיום התהליך לפני שהוא מגיע לסיומו הטבעי) רק את מכונת DOS היחידה שמריצה את אותו תהליך; כל תוכנית אחרת כלשהי שמופעלת במכונת DOS שעדיין פועלת, ממשיכה בפעולתה. ליישומי Windows מבוססי 16 סיביות ש-Windows מפעילה מכונות DOS נפרדות יש **תורי קלט** (Input Queues) נפרדים. על כן, אם אחד מהיישומים נעצר, שאר היישומים האחרים, הפועלים במכונות DOS אחרות, ממשיכים לקבל קלט.
CREATE_SHARED_WOW_VDM	Windows NT: הדגל תקף רק כאשר מתחילים ביישום Windows מבוסס 16 סיביות. אם המפתח DefaultSeparateVDM שבמקטע או בקבוצת Windows של הקובץ WIN.INI שווה ל-True, דגל זה גורם לפונקציה CreateProcess **לעקוף** (Override) את המפתח, ולהפעיל את התהליך החדש ב-VDM המשותף.
CREATE_SUSPENDED	Windows יוצרת את המטלה הראשית של התהליך החדש ב**מצב מושהה** (Suspended State) ואינה מפעילה את התהליך החדש עד שמטלה אחרת (בתהליך אחר) קוראת לפונקציה ResumeThread.
CREATE_UNICODE_ENVIRONMENT	אם קובעים דגל זה, בלוק הסביבה lpEnvironment משתמש בתווי Unicode. אם לא קובעים דגל זה, בלוק הסביבה משתמש בתווי ANSI.
DEBUG_PROCESS	אם קובעים דגל זה, Windows מתייחסת לתהליך הקורא כ**תוכנית ניפוי** (Debugger), תוכנית שירות, הכלולה לעיתים במהדרים או בפרשנים, ומסייעת לתוכניתנים למצוא ולתקן שגיאות תחביר ושגיאות אחרות בקוד המקור) והתהליך החדש הופך להיות עוד תהליך שתוכנית הניפוי של התהליך הקורא מנפה. המערכת מודיעה לתוכנית הניפוי על

דגל	פירוש
DEBUG_PROCESS	כל **אירועי הניפוי** (Debug Events) שמתרחשים בתהליך שנמצא בבדיקה. אם יוצרים תהליך וקובעים את הדגל הזה, רק התהליך הקורא (זה שקרא ל-CreateProcess) יכול לקרוא לפונקציה WaitForDebugEvent.
DEBUG_ONLY_THIS_PROCESS	אם לא קובעים דגל זה ו-Windows מנפה כרגע את התהליך הקורא, התהליך החדש הופך להיות עוד תהליך שמנופה על ידי המנפה של התהליך הקורא. אם התהליך הקורא אינו תהליך ש-Windows מנפה כרגע, לא מתרחשות פעולות ניפוי כלשהן כתוצאה מדגל זה.
DETACHED_PROCESS	עבור תהליכים מסוג קונסול, אין לתהליך החדש גישה לקונסול של תהליך האב. התהליך החדש יכול לקרוא לפונקציה AllocConsole אחר כך כדי ליצור קונסול חדש. אי אפשר להשתמש בדגל זה עם הדגל CREATE_NEW_CONSOLE.

כבר למדת שאפשר לבחור דגל אחד או יותר עבור הפרמטר dwFlags. אך אפשר להגדיר רק דגל עדיפות מחלקה אחד לפרמטר dwFlags. דגל עדיפות המחלקה חייב להיות אחד הערכים שמפורטים בטבלה 16.3.

טבלה 16.3: דגלי העדיפות האפשריים של הפונקציה **CreateProcess**.

דגל	פירוש
HIGH_PRIORITY_CLASS	מציין תהליך שמבצע **משימות זמן קריטי** (Time-Criticai Tasks) ש-Windows חייבת להפעיל מיד עבור התהליך, כדי שיפעל נכון. מטלות של תהליכי **מחלקה בעדיפות גבוהה** (High-Priority Class) יוצרות פסיקה למטלות שנובעות מתהליכי **מחלקה בעדיפות נורמלית** (Normal-Priority Class), או **עדיפות המתנה** (Idle-Priority), ולכן גם מתבצעות מיד. לדוגמה, **רשימת המשימות** (Task List) של Windows חייבת להגיב במהירות כאשר המשתמש קורא לה, ללא תלות בעומס על מערכת ההפעלה. חייבים לנקוט בזהירות רבה כאשר משתמשים בקביעת מחלקה בעדיפות גבוהה, מכיון שקביעת עדיפות כזו ליישום שצורך זמן מעבד רב, עלולה לגרום לחסימת המעבד מפני יישומים אחרים.

דגל	פירוש
HIGH_PRIORITY_CLASS	**מצב המתנה** - **idle state** - מציין מצב של אובייקט, או התקן שניתן להפעלה, אך אינו בשימוש, או שהוא ממתין לפקודה שתורה לו להתחיל לפעול.
IDLE_PRIORITY_CLASS	מציין תהליך שהמטלות שלו מופעלות רק כאשר המערכת במצב **המתנה** (Idle) ומטלות של תהליך אחר כלשהו במחלקה בעדיפות גבוהה, יכול לקבל שירות לפניו, כמו לדוגמה, שומר מסך. תהליכי בן יורשים את המחלקה בעדיפות המתנה.
NORMAL_PRIORITY_CLASS	מציין תהליך נורמלי בלי רשימת צרכים מיוחדת.
REALTIME_PRIORITY_CLASS	מציין תהליך שנקבעה לו העדיפות הגבוהה ביותר האפשרית. המטלות במחלקה בעדיפות זמן אמת (Real-Time Priority Class) מקבלים שירות **לפני** כל שאר התהליכים, ובכלל זה תהליכים של מערכת ההפעלה שמבצעים משימות חשובות. לדוגמה, תהליך זמן אמת שמופעל יותר מאשר לפרק זמן קצר מאוד עלול לכך שמטמון הדיסק לא יפעל, או שהעכבר לא יגיב.

בנוסף ליצירת תהליך, CreateProcess יוצרת גם **אובייקט מטלה**. הפונקציה CreateProcess יוצרת את המטלה עם מחסנית התחלתית שגודלה מוצג **בכותרת התמונה** (Image Header) של קובץ ההפעלה של התוכנית המוגדרת. המטלה מתחילה להתבצע מנקודת הכניסה המוגדרת בכותרת זו. CreateProcess יוצרת את התהליך החדש ואת ידיות המטלות החדשות עם זכויות גישה מלאות. עבור שתי הידיות, אם הפונקציה אינה מספקת **מתאר אבטחה** (Security Descriptor), התוכניות יכולות להשתמש בידית בכל פונקציה שדורשת ידית לאותו סוג של אובייקט. כאשר הפונקציה מספקת מתאר אבטחה, התוכניות מבצעות בדיקת גישה בכל השימושים העוקבים של הידית לפני שהם מורשים לגשת למטלה. אם בדיקת הגישה דוחה את אפשרות הגישה, התהליך הדורש אינו יכול להשתמש בידית לקבלת גישה למטלה.

Windows מציבה בתהליך מזהה בן 32 סיביות. המזהה תקף עד שהתהליך מסתיים. התוכניות יכולות להשתמש במזהה זה כדי לזהות את התהליך או להגדיר אותו עבור הפונקציה OpenProcess לפתיחת ידית לתהליך. Windows מציבה למטלה ההתחלתית של התהליך מזהה בן 32 סיביות. המזהה תקף עד שהמטלה מסתיימת והתוכנית יכולה להשתמש במזהה זה כדי לזהות במפורש את המטלה במערכת. מזהים אלה מוחזרים במבנה PROCESS_INFORMATION.

כאשר אתה מגדיר שם יישום במחרוזות lpApplicationName או lpCommandLine, אין כל צורך ששם הקובץ של היישום יכיל סיומת, אלא רק במקרה של יישום מבוסס MS-DOS או מבוסס Windows, שסיומת שם הקובץ שלו היא COM. המטלה הקוראת יכולה להשתמש בפונקציה WaitForInputIdle כדי לחכות עד שהתהליך החדש מסיים את האתחול שלו ומחכה לקלט מהמשתמש, בלי שיהיה קלט ממתין. השימוש ב-WaitForInputIdle יכול להיות שימושי עבור הסינכרון בין תהליך האב לבין תהליך הבן, מכיון ש-CreateProcess חוזרת מבלי לחכות שהתהליך החדש יסיים את האתחול שלו. לדוגמה, התהליך היוצר צריך להשתמש ב-WaitForInputIdle לפני שהוא מנסה למצוא חלון שקשור לתהליך החדש.

הדרך המומלצת של מיקרוסופט לסגירת תהליכים היא באמצעות הפונקציה ExitProcess, מכיון שהיא מודיעה לכל תיקיות הקישור הדינמי (DLLs) שצמודות לתהליך על כוונת הסיום. אמצעים אחרים לסגירת תהליך אינם עושים זאת. שים לב, כאשר מטלה קוראת ל-ExitProcess, הפונקציה מסיימת את שאר מטלות התהליך מבלי לתת להן הזדמנות להפעיל קטע קוד נוסף כלשהו (כולל קוד הסיום של המטלה של תיקיות קישור דינמי מוצמדות).

Windows **מפעילה ברצף** (Serializes) את ExitProcess, ExitThread, CreateThread, CreateRemoteThread, ותהליך שמתחיל (כתוצאה מקריאה ל-CreateProcess) ביניהם בתוך התהליך. מאורעות אלה יכולים לקרות בכל זמן נתון במרחב הכתובות, ופירוש הדבר שחלות הגבלות אלו:

- במהלך התחלת ההפעלה של התהליך ואתחול שגרות תיקיית הקישור הדינמי, התוכניות יכולה ליצור מטלות חדשות, אבל הן אינן מתחילות להתבצע עד ש-Windows מסיימת את אתחול תיקיית הקישור הדינמי של התהליך.
- מטלה אחת בלבד יכולה לפעול במהלך האתחול של תיקיית הקישור הדינמי, או **בשיגרת ניתוק** (Detach Routine) בכל זמן נתון.
- הפונקציה ExitProcess אינה חוזרת, כל עוד יש מטלה פעילה באתחול תיקיית קישור דינמי, או שגרות ניתוק.

התהליך הנוצר נשאר במערכת עד שכל מטלות התהליך הסתיימו והקריאה לפונקציה ClosseHandle סגרה את כל הידיות של התהליך ואת המטלות שלו. הקריאה ל-ClosseHandle חייבת לסגור את הידיות אל התהליך ואל המטלה הראשית. אם התוכנית אינה זקוקה לידיות אלו, עדיף לסגור אותן מייד לאחר ש-Windows יוצרת את התהליך.

כאשר מטלה אחרונה בתהליך מסתיימת, מתרחשים האירועים האלה:

- Windows סוגרת באופן בטוח את כל האובייקטים שנפתחו על ידי התהליך.
- מצב הסיום של התהליך (שמוחזר על ידי GetExitCodeProcess) משתנה מערכו ההתחלתי STILL_ACTIVE למצב הסיום של המטלה האחרונה שמסתיימת.

☆ Windows קובעת את אובייקט המטלה של המטלה הראשית למצב אות (Signaled State). כך היא מספקת את כל המטלות שחיכו לאובייקט.

☆ Windows קובעת את אובייקט התהליך למצב אות, וכך מספקת את כל המטלות שחיכו לאובייקט.

אם התיקיה הנוכחית בכונן C היא CBIBLE\BIBLE\, יש משתנה סביבה בשם =C, שהערך שלו הוא C:\CBIBLE\BIBLE. כמו שהוסבר קודם בעניין lpEnvironment, מידע כזה על התיקיה הנוכחית של כונן המערכת אינו ידוע אוטומטית לתהליך חדש כאשר הפרמטר lpEnvironment של הפונקציה CreateProcess אינו NULL. היישום צריך להעביר ידנית את המידע על התיקיה הנוכחית לתהליך החדש. כדי לעשות זאת, היישום חייב ליצור בצורה ברורה את מחרוזות משתנה הסביבה =X, לפי סדר האלף בית (מכיון ש- Windows NT ו- Windows 9x משתמשות בסביבה ממוינת), ואחר כך לשים אותן בבלוק הסביבה, כפי שהוסבר קודם בעניין אחסון הבלוק.

אחת הדרכים להשגת המשתנה של התיקיה הנוכחית עבור כונן X היא לקרוא לפונקציה GetFullPathName, (. .,"X:"), שמאפשרת ליישום להימנע מסריקת בלוק הסביבה. אם המסלול המלא המוחזר על ידי GetFullPathName הוא \:X, התוכנית אינה צריכה להעביר את הערך הזה הלאה כמידע סביבה, מכיון שתיקיית השורש היא תיקיית ברירת המחדל הנוכחית עבור כונן X של התהליך חדש. הפונקציה CreateProcess מחזירה את הידית שיש לה גישת PROCESS_ALL_ACCESS אל אובייקט התהליך.

התיקיה הנוכחית המוגדרת על ידי הפרמטר lpCurrentDirectory היא התיקיה הנוכחית של תהליך הבן. התיקיה הנוכחית בפרמטר lpCommandLine היא התיקיה הנוכחית של תהליך האב.

<u>שים לב:</u>

במערכת Windows NT, כשפונקציה יוצרת תהליך בשעה שמוגדר דגל העדיפות CREATE_NEW_PROCESS_GROUP, מערכת ההפעלה מבצעת קריאה בטוחה ל- SetConsoleCtrlHandler(NULL,True) עבור התהליך החדש; כלומר, התהליך החדש כולל אות שינוי למצב לא פעיל, הנובע מצירוף המקשים Ctrl+C. הדבר מאפשר למעטפות (Shells) טובות לטפל בצירוף המקשים Ctrl+C בעצמן, ולהעביר אות זה לתת-תהליכים באופן מבוקר. Ctrl+Break אינו אות מעבר למצב לא-פעיל, ו-Windows NT יכולה אולי להשתמש בצירוף זה כדי להפסיק את התהליך ואת קבוצת התהליך.

16.3 סיום תהליכים

למדת שתוכניות מופעלות במסגרת תהליך ובסעיף 16.2 למדת איך יוצרים תהליך. כמו במרבית ההקצאות או ביצירת עצמים בתוכניות, זוהי אחריות המתכנת לצאת מתהליך שסיים את תפקידו. משתמשים בפונקציה ExitProcess כדי לסגור את התהליך שמתבצע כרגע. הפונקציה ExitProcess מסיימת את התהליך ואת כל המטלות הקשורות בו וחוזרת למקום שממנו קראו לה, כמו שרואים להלן:

```
VOID ExitProcess(
   UINT uExitCode              // exit code for all threads
);
```

הפרמטר uExitCode מגדיר את קוד היציאה מהתהליך ומכל המטלות שמסתיימות על ידי התהליך כתוצאה מהקריאה לפונקציה ExitProcess. משתמשים בפונקציה GetExitCodeProcess כדי לקבל את ערך היציאה של התהליך, ובפונקציה GetExitCodeThread משתמשים כדי לקבל את ערך היציאה של המטלה.

צריך תמיד לקרוא ל-ExitProcess כדי לסיים תהליך. הפונקציה ExitProcess מספקת סגירה חלקה ונקיה של התהליך, אשר כוללת את הקריאה ל**פונקציית נקודת הכניסה** (Entry-Point Function) של כל **תיקיות הקישור הדינמי** (DLLs) הצמודות, עם ערך שמציין שהתהליך מסיים את ההצמדה אל תיקיית הקישור הדינמי. אם תהליך קורא ל-TerminateProcess כדי לסיים את פעולתו, הוא אינו מודיע על כך לתיקיות הקישור הדינמי ש-Windows אולי הצמידה לתהליך.

לאחר שכל תיקיות הקישור הדינמי עיבדו ערך סיום כלשהו של התהליך, הפונקציה ExitProcess מסיימת את התהליך הנוכחי. סיום תהליך כרוך בפעולות הבאות:

1. הפונקציה ExitProcess סוגרת את כל ידיות האובייקט שהתהליך פתח.
2. כל המטלות של התהליך מסיימות את פעולתן.
3. המצב של אובייקט התהליך הופך ל**מסומן** (Signaled), והדבר גורם **להפעיל מחדש** (Resume) מטלות כלשהן שהמתינו לתהליך שיסתיים.
4. המצב של כל מטלות התהליך הופך **למסומן**, והדבר גורם להפעלה מחדש של מטלות כלשהן שהמתינו לתהליך שיסתיים.
5. מצב הסיום של התהליך משתנה מ-STILL_ACTIVE לערך של קוד היציאה של התהליך.

סיום תהליך אינו גורם ל-Windows לסיים את תהליכי הבן. סיום תהליך אינו גורם בהכרח להסרת אובייקט התהליך ממערכת ההפעלה. המערכת מוחקת אובייקט תהליך כאשר התוכנית סוגרת את הידית האחרונה של התהליך. Windows מפעילה באופן סדרתי את הפונקציות ExitProcess, ExitThread, CreateThread, ו-CreateRemoteThread.

בנוסף לכך, Windows מפעילה באופן סדרתי תהליך שמתחיל (כתוצאה מקריאה ל-CreateProcess) עם התהליכים הקודמים שבתוך התהליך הקורא. רק אירוע אחד מתוך אירועים אלה יכול לקרות במרחב הכתובת בכל זמן נתון; כלומר, התהליך מקיים את המגבלות הבאות:

- ✰ בזמן התחלת הפעלת התהליך ושגרות האתחול של תיקיות הקישור הדינמי, התוכנית יכולה ליצור מטלות חדשות, אבל הן אינן מתחילות לפעול עד שמסתיים הליך האתחול של תיקיות הקישור הדינמי עבור תהליך זה.
- ✰ רק מטלה אחת בתהליך יכולה להיות בכל זמן נתון באתחול של תיקיית קישור דינמי, או בשגרה מנתקת (Detach Routine).
- ✰ ExitProcess אינה חוזרת עד שאף לא אחת מהמטלות נמצאת באתחול תיקיית הקישור הדינמי שלהן, או בשגרות מנתקות.

16.4 יצירת תהליכים - תהליכי בן

למדת שאפשר להשתמש בפונקציה CreateProcess כדי להתחיל בהפעלת תהליך אחר. כאשר אתה מתכנן יישומים מורכבים יותר, תפגוש פעמים רבות במצבים שבהם תרצה לבצע בלוק קוד אחר למילוי משימה כלשהי. אפשר לקרוא מהתוכנית לפונקציה שתבצע את המשימה. עם זאת, פונקציות הן סדרתיות בטבען, ומשמעות הדבר היא שהקוד אינו יכול להמשיך בביצוע עד לאחר סיום פעולת הפונקציה.

תחליף לשיטה שבה משתמשים בבלוק אחר של קוד לביצוע המשימות השונות בתוכניות הוא ליצור מטלה חדשה בתהליך, ולתת לה לסייע בהמשך העיבוד. השימוש ב**ריבוי מטלות** (Multiple Threads) מאפשר לקוד התוכנית להמשיך בעיבוד בזמן שהמטלה החדשה מבצעת את המשימה הנדרשת ממנה. לרוע המזל, השימוש בריבוי מטלות פעמים רבות גורם **לבעיות תזמון** (Synchronization Problems), כאשר חובה על המטלה הקיימת "לראות" את תוצאות המטלה החדשה. על תזמון תלמד בסעיפים הבאים.

הגישה האחרת היא יצירת תהליך חדש, שנקרא **תהליך בן** (Child Process), שתפקידו לתמוך באפשרות הביצוע של פעולות במקביל. תהליכים מאפשרים לתוכניות להמשיך במסלול העיבוד העיקרי שלהן, בשעה שהן מטילות על תהליך בן לבצע תהליך עיבוד משני כלשהו. אפשר גם לעכב את פעולת התוכנית בשעה שתהליך הבן פועל במטלה שתוצאותיה דרושות להמשך פעולתו של התהליך העיקרי. בסעיפים הבאים נעסוק בתהליכי בן ובמטלות. השימוש בשני כלים חשובים אלה יעזור לך להבין טוב יותר את ההבדלים ביניהם, ואיך אפשר להשתמש בשניהם בתוכניות.

ככל שתוכניות Windows שלך נעשות מורכבות יותר, תשתמש יותר ויותר ברכיבים שאין תלות ביניהם, וכאלה שהם משותפים. רכיבים אלה יכולים להיות, לדוגמה, תיקיות קישור דינמי, שפועלות מתוך התהליך הנוכחי של התוכנית (אובייקט פנים תהליכי, In-Process Objects); או שרתים אוטומטיים (Automation Servers) לקישור

והטבעה של אובייקטים (Object Linking And Embedding - OLE), שפועלים מחוץ לתהליך הנוכחי של התוכנית (אובייקט חוץ תהליכי, Out-Of-Process Objects). ספר זה לא דן הרחבה באובייקטים מסוג in-process או באובייקטים מסוג out-of-process.

16.5 עובדים יותר עם תהליכי בן

כמו שלמדת, התוכניות יכולות לשלוט בדרך שבה הן פועלות עם תהליכי בן. אך כמעט כל תהליכי הבן דורשים גישה לנתונים המוכלים במרחב הכתובת של תהליך האב. ככלל, כאשר תהליך בן מבקש גישה לנתונים של תהליך האב, צריך להריץ את תהליך הבן במרחב כתובות משלו, אך לאפשר לו גישה לנתונים הרלוונטיים במרחב הכתובת של תהליך האב. השליטה בגישה למרחב הכתובת של תהליך האב מאפשרת לך להגן על נתונים שאינם רלוונטיים לתהליך הבן מפני פגיעה לא מכוונת. Win32 מאפשרת מספר שיטות להעברת נתונים בין תהליכים שונים: **חילוף נתונים דינמי** (DDE - Dynamic-Data Exchange), **קישור והטבעה של אובייקטים** (Object Linking And - OLE Embedding), **צינורות** (Pipes), **חריצי דואר** (Mail Slots), וכדומה. אחת הדרכים המקובלות ביותר, וגם הפשוטה ביותר, להשתתף בנתונים היא להשתמש ב**קובץ ממופה-זיכרון** (Memory-Mapped File).

קובץ ממופה-זיכרון הוא קובץ מסוג מיוחד, שמאפשר **לשמור אזור של מרחב כתובות** (Reserve A Region Of Address Space) ולשריין אמצעי אחסנה פיסי עבור האזור הזה, בדומה להקצאת זיכרון וירטואלית. אך לא כמו הזיכרון הווירטואלי, בקבצים הממופים בזיכרון אמצעי האחסנה הפיסי בא מקובץ שקיים כבר בדיסק, ולא מ**קובץ הדפדוף** (Paging File) של המערכת. אחרי מיפוי הקובץ, אפשר לגשת לכל חלקיו, כאילו התוכנית טענה אותו לזיכרון.

משתמשים בקבצים הממופים האלה עבור שלוש המטרות שלהלן:

- ✰ המערכת משתמשת בקבצים ממופי-זיכרון כדי לטעון ולהפעיל קבצי הפעלה וקבצי תיקיות קישור דינמי. השימוש בקבצים ממופים חוסך במקום שצריך עבור קובץ הדפדוף, וגם בזמן שהיישום צריך כדי להתחיל לפעול.
- ✰ אפשר להשתמש בקבצים ממופי-זיכרון כדי לגשת לקבצי נתונים בדיסק. השימוש בקבצים הממופים חוסך ביצוע פעולות קלט ופלט בקבצים ושימוש בחוצצים להעברת תכולת הקובץ שפועלים עליו.
- ✰ אפשר להשתמש בקבצים הממופים כדי לאפשר לתהליכים רבים שפועלים באותה מחשב לשתף נתונים ביניהם, כמו שלמדת עד עכשיו.

רבים מאובייקטי התקשורת של Win32 משתמשים במבנה קבצים ממופי זיכרון, מכיון שזהו כלי חזק וקל לשימוש. בסעיפים שתלמד מאוחר יותר תלמד כיצד ליצור קבצים ממופי זיכרון.

אם רוצים ליצור תהליך חדש, וגם רוצים שיבצע עבודה כלשהי בזמן שתהליך האב ממתין לתוצאה (לדוגמה, לכתוב קטע נתונים לקובץ שתהליך האב יקרא יותר מאוחר), תהליך האב יכול להשתמש בקוד שדומה לקוד שלהלן:

```
PROCESS_INFORMATION  pi;
DWORD dwExitCode;

BOOL fSuccess = CreateProcess(ProcessName, &pi);
if (fSuccess)
  {
   // close the thread handle as soon as you no longer need it
   CloseHandle(pi.hThread);
   WaitForSingleObject(pi.hProcess, INFINITE);

   // The process terminated
   GetExitCodeProcess(pi.hProcess, &dwExitCode);

   // close the process handle
   CloseHandle(pi.hProcess);
  }
```

קטע הקוד הזה יוצר את התהליך החדש ProcessName. אם הוא מצליח, קטע הקוד סוגר את הידית המיותרת כדי לשחרר זמן CPU, ואחר כך ממתין לתהליך שישלים את העיבוד שלו. לאחר שהתהליך משלים את העיבוד, המשתנה dwExitCode מכיל את המידע על קוד היציאה של התהליך, וקטע הקוד סוגר את ידית התהליך. אם היה כישלון בטעינת התהליך, קטע הקוד אינו מבצע עיבוד כלשהו.

16.6 הפעלת תהליך בן מנותק (Detached Child Process)

בסעיף 16.5 למדת ליצור תהליך בן וגם להפסיק (Halt) את הפעלת המטלות הנוכחיות עד שהתהליך מסתיים. במקרים רבים התוכניות מתחילות בתהליך אחר כ**תהליך בן מנותק** (Detached Child Process), שאינו מוצמד לתהליך האב. תהליך בן מנותק הוא תהליך אשר תהליך האב יוצר בזמן הפעלתו, ולאחר מכן, כשתהליך הבן מתחיל להתבצע, תהליך האב אינו מעוניין בתקשורת כלשהי עם תהליך הבן, או שתהליך האב אינו דורש שתהליך הבן יסיים את עבודתו, כדי שהוא עצמו יוכל להמשיך לפעול. הרצת תהליכי בן מנותקים מאפשרת לתוכניות לטעון תוכניות אחרות מבלי לדאוג לזמן הניהול שלהן או למהירות הביצוע שלהן. כאשר מפעילים תוכנית **סייר Windows** (Windows Explorer) לדוגמה, הסייר יוצר תהליך חדש, ואחר כך מתעלם מהתהליך החדש וממשיך במה שנדרש ממנו לעשות.

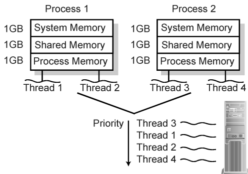
כאשר התוכניות יוצרות תהליך בן מנותק, התוכנית חייבת ליצור תחילה את התהליך, ואחר כך לסגור את ידיות התהליך החדש ואת המטלה הראשית שלו. קטע הקוד שלהלן מציג כיצד אפשר ליצור תהליך בן חדש מנותק, שאינו מוצמד:

```
BOOL fSuccess = CreateProcess(ProcessName, &pi);
if (fSuccess)
  {
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);
  }
```

16.7 הבנה מעמיקה יותר של המטלות

בסעיפים קודמים למדת ליצור ולנהל תהליכים. כל תהליך של Win32 מכיל מטלה אחת או יותר. ראוי לזכור שמטלה היא נתיב פעילות בתוך תהליך. בכל פעם ש-Windows מאתחלת מופע חדש של תהליך, מערכת ההפעלה יוצרת מטלה ראשית חדשה עבורו. המטלה הראשית מתחילה כאשר Windows טוענת את התוכנית. המטלה, בתורה, קוראת לפונקציה WinMain וממשיכה בפעולתה עד שהפונקציה WinMain מסיימת את העיבוד שלה (סיום העיבוד של המטלה) והתוכנית קוראת ל-ExitProcess כדי שיסיים את התהליך. עבור הרבה יישומים, המטלה הראשית שיוצרת מערכת ההפעלה היא המטלה היחידה שהתוכנית צריכה. עם זאת, התהליכים יכולים ליצור מטלות נוספות כדי לעזור להם בביצוע משימותיהם. הרעיון שעומד מאחורי יצירת מטלות נוספות הוא להשתמש בכמה שיותר זמן מעבד וביעילות מירבית. כאשר אתה יוצר מטלות נוספות, אתה שולח למערכת ההפעלה דרישות נוספות לקבלת זמן מעבד. כמו שתלמד בהמשך, מטלות נוספות מאפשרות לתוכניות לבצע ביתר יעילות עיבוד ברקע, לבצע חישובים מורכבים, ופעולות מבוססות זמן ואירוע (Time And Event Based Activities), כמו גם ביצוע משימות תכנות מתקדמות. בסעיפים הבאים נדון מתי ליצור ומתי לא ליצור מטלה.

כאשר אתה חושב מה מופעל כרגע במערכת, תוכל להעזר בכך שתחשוב על המטלות כארוזות בתוך תהליך. המערכת מאפשרת למטלות השונות לפעול ולחזור ולפעול, כתלות ב**עדיפות** (Priority) כל מטלה, וללא קשר אם הן שייכות לתהליך אחד או לתהליכים שונים. תרשים 16.4 מציג מודל לוגי של מספר תהליכים, שכל אחד מהם מכיל מטלות רבות שפועלות במערכת ההפעלה Windows, כמו כן סדר אפשרי להקצאת זמן מעבד עבורן.



תרשים 16.4: המודל הלוגי של תהליך-מטלה.

16.8 הערכת הצורך במטלות

בסעיף קודם למדת שהתוכניות משתמשות פעמים רבות במטלות כדי להבטיח שלחלקי היישום השונים תהיה גישה למשאבי המעבד. ההחלטה מתי להשתמש במטלות נוספות ומתי לא להשתמש בהן, היא פעמים רבות ההחלטה החשובה ביותר שעושים בעת תכנות בסביבת Windows.

לדוגמה, התבונן בתוכנית גיליון אלקטרוני. תוכנית זו חייבת לבצע חישובים חוזרים כאשר המשתמש משנה את הנתונים שבתוך התאים. החישובים החוזרים לגליונות אלקטרוניים מורכבים לעיתים ודרוש זמן מה כדי לבצעם. על כן, יישום המתוכנן היטב אינו צריך לבצע חישוב חוזר לגיליון האלקטרוני לאחר כל שינוי שעושה המשתמש. במקום זאת, עליו להפעיל את החישוב החוזר של הגיליון כמטלה נפרדת, בעלת עדיפות יותר נמוכה מזו שיש למטלה הראשית. כאשר משתמשים בשתי מטלות, המטלה הראשית תמיד תפעל בזמן שהמשתמש מקליד, ואז מטלת החישוב החוזר בעלת העדיפות הנמוכה לא תוכל לגשת למעבד. כאשר המשתמש מפסיק להקליד, המטלה בעלת העדיפות הנמוכה מתחילה להתבצע, בעוד המטלה הראשית ממתינה למשתמש שיתחיל שוב בהקלדה.

אפשר ליישם את העיקרון שלמדת זה עתה, של השימוש במטלות, גם עבור תוכניות מורכבות שמבצעות קבוצות של משימות רבות. פעילויות ברקע גם כן מפעילות מטלות נוספות.

לפניך מספר מצבים נוספים שבהם יש תועלת משימוש במטלות:

- ✫ יצירת מטלה נפרדת לביצוע משימות ההדפסה של היישום, כדי לאפשר למשתמש להמשיך ולהפעיל את היישום עצמו בזמן ההדפסה.
- ✫ יצירת מטלה נפרדת כדי להחזיק **תיבת דו-שיח לא מודאלית** (Modeless Dialog Box) שמאפשרת למשתמש להפסיק (Interrupt) משימות שנמשכות זמן רב, כמו העתקת קבצים או הדפסה.
- ✫ יצירת מטלות שונות הפועלות במקביל ובו-זמנית כדי לדמות את אירועי העולם הממשי (לדוגמה, אירועים שמתרחשים בפרקי זמן שאינם מוגדרים).

16.9 ההחלטה לא להפעיל מטלה

כשמתכנתים פועלים לראשונה בסביבת עבודה שתומכת בריבוי מטלות, הם נוטים לשימוש יתר במטלות, בגלל העוצמה של שיטת העיבוד החדשה ונוחות השימוש שהמטלות מאפשרות להם. מתכנתים רבים התחילו לחלק יישומים קיימים לחלקים קטנים יותר, שכל אחד מהם מופעל כמטלה נפרדת. למרות נוחות השימוש ב**עיבוד מבוסס מטלות** (Thread-Based Processing) ועד כמה שהוא נראה יעיל ממבט ראשון, הוא גם יכול לגרום לבזבוז משאבי זמן עיבוד של היישום. יצירת מטלות והפעלתן דורשת תקורה כלשהי, כמו למשל ניהול תורי מטלות נפרדים ומעקב אחריהן, גם אם מערכת ההפעלה אינה משחררת את המטלה לעיבוד. מטלות רבות מאיטות את ביצועי המערכת, מכיון שבנוסף לתקורת המטלה המערכת חייבת לבדוק את **רמת העדיפות** (Priority Level) של כל מטלה פעילה כרגע כאשר עליה להחליט איזו מטלה להפעיל.

כמו שלמדת, נוח להשתמש במטלות ויש להן גם יתרונות ולכן - יש להן מקום בכל תוכניות Windows. אך חשוב לדעת, שכאשר משתמשים במטלות, עלולות להיווצר בעיות שונות, בעת ניסיון לפתור בעיות הפעלה וביצועים. לדוגמה, כשאתה מפתח תוכנית עיבוד תמלילים ורוצה שפעולת ההדפסה תפעל במטלה נפרדת, התגובה הראשונה שלך יכולה להיות ליצור מטלה ולהפעילה להדפסת המסמך דף אחרי דף. לרוע המזל, המשתמש יכול לשנות את המסמך בזמן שאתה מדפיס ברקע. על כן, במקום הפתרון הפשוט הקודם, עליך להעתיק את הקובץ המיועד להדפסה אל קובץ זמני, להדפיס אותו ולמחוק אותו בסיום ההדפסה. השימוש בריבוי מטלות ובקובץ זמני הוא בלי ספק יותר יעיל למשתמש וגם הופך את התוכנית ליותר מתאימה עבורו. עם זאת, עליך להיות זהיר כדי לוודא שיצירת מטלות נוספות אינה מסכנת את העיבוד שמבצעת התוכנית שלך באמצעות המטלות הנוכחיות שלה.

מספר כללים צריכים להתקיים כאשר מחליטים להשתמש, או לא להשתמש, **בריבוי מטלות** (Multiple Threads):

- ✫ כל מרכיבי הממשק למשתמש (פקדים וחלונות) צריכים להשתתף במטלה משותפת, עם חריגות נדירות מאוד.
- ✫ תוכניות צריכות ליצור מטלות רק כשצריכים אותן, ולא ליצור מטלות נוספות ו"להחזיקן במלאי".

- התוכניות צריכות לשחרר מטלות כשפעולתן מסתיימת.
- יישומים מורכבים יותר, אשר משתמשים בחלונות רבים, עשויים לגרור יצירת מטלות נוספות כדי לבצע את העיבוד בחלונות מסוימים.
- לא צריך ליצור מטלות שעלולות לאפשר למשתמש להפריע לתהליכי מערכת קריטיים, דבר שעלול לגרום לשיבוש בזיכרון או התהליכים אחרים שפועלים באותו זמן.

ככלל, עליך להיות תמיד בטוח שאתה זקוק למטלה, או שהמטלה משפרת בצורה משמעותית את תהליכי העיבוד של התוכנית. השימוש במטלות נוספות מבלי לעשות את השיקולים האלה, גורם לבזבוז **זמן מעבד** (Processor Time) ולהאטה בהפעלת התוכניות.

16.10 יצירת פונקציית מטלה פשוטה

בסעיפים קודמים למדת שהתוכניות יכולות להשתמש בריבוי מטלות כדי לאפשר להן לבצע **פעולות עיבוד מרובב** (Multiple Execution Activities) במסגרת תהליך יחיד. כאשר יוצרים מטלות, התוכניות משתמשות בדרך כלל בפונקציה CreateThread. פונקציה זו יוצרת מטלה שתופעל במרחב הכתובות של התהליך הקורא. את הפונקציה CreateThread כותבים בתוכנית, כמו שרואים בהגדרה שלהלן:

```
HANDLE CreateThread(
 LPSECURITY_ATTRIBUTES lpThreadAttributes,
                          // ptr to thread security attributes
 DWORD dwStackSize,       // initial thread stack size, in bytes
 LPTHREAD_START_ROUTINE lpStartAddress,
                          // pointer to thread function
 LPVOID lpParameter,      // argument for new thread
 DWORD dwCreationFlags,   // creation flags
 LPDWORD lpThreadId       // ptr to returned thread identifier
);
```

הפונקציה CreateThread מקבלת את הפרמטרים שמפורטים בטבלה 16.4.

טבלה 16.4: הפרמטרים שמקבלת הפונקציה **CreateThread**.

פרמטר	תיאור
lpThreadAttributes	מצביע למבנה מסוג SECURITY_ATTRIBUTES שקובע אם תהליך הבן יכול לרשת את הידית המוחזרת. אם lpThreadAttributes הוא NULL, תהליך הבן אינו יכול לרשת את הידית. תחת Windows NT, האיבר lpSecurityDescriptor של המבנה מגדיר **מתאר אבטחה** (Security Descriptor) למטלה החדשה. אם lpThreadAttributes הוא NULL, המטלה

פרמטר	תיאור
lpThreadAttributes	מקבלת את ברירת המחדל של מתאר האבטחה. תחת Window9x, הפונקציה CreateThread מתעלמת מהאיבר lpSecurityDescriptor של המבנה.
dwStackSize	מגדיר את הגודל, בבתים, של מחסנית המטלה החדשה. אם dwStackSize שווה לאפס, גודל המחסנית הוא לפי ערך ברירת המחדל; זהו גודל המחסנית של המטלה הראשית של התהליך. Windows מקצה אוטומטית את מחסנית המטלה במרחב הזיכרון של התהליך, ומשחררת את המחסנית כאשר המטלה מסתיימת. שים לב לכך שגודל המחסנית משתנה, אם יש צורך בכך. CreateThread מנסה לשריין את מספר הבתים שמוגדרים על ידי dwStackSize, והיא נכשלת אם הגודל עובר את הזיכרון הזמין.
lpStartAddress	כתובת ההתחלה של המטלה החדשה. ככלל, זו הכתובת של פונקציה שהוכרז עליה על פי מוסכמות נוהל הקריאה WINAPI. לפיו, הממשק הזה מקבל מצביע בודד בן 32 סיביות כארגומנט, ומחזיר ערך קוד יציאה בן 32 סיביות. ההגדרה נכתבת כך: DWORD WINAPI ThreadFunc(LPVOID);
lpParameter	מגדיר ערך בודד לפרמטר בן 32 סיביות שמועבר למטלה.
dwCreationFlags	מגדיר דגלים נוספים ששולטים ביצירת המטלה. אם dwCreationFlags מגדיר את הדגל CREATE_SUSPENDED, התהליך יוצר את המטלה **במצב מושהה** (Suspended State), ואינו מפעיל אותה עד שהתוכנית (או תוכנית אחרת) קוראת לפונקציה ResumeThread. אם הפרמטר שווה לאפס, המטלה מופעלת מיד לאחר שהיא נוצרת. Windows אינה תומכת כרגע בערכים אחרים.
lpThreadId	מצביע למשתנה בן 32 סיביות שמקבל את המזהה של המטלה.

אם הפונקציה CreateThread מצליחה, הערך המוחזר הוא ידית למטלה החדשה. אם הפונקציה נכשלת, הערך המוחזר הוא NULL. תחת Window 9x, הפונקציה CreateThread מצליחה רק כאשר המערכת קוראת לה בהקשר של תוכנית 32 סיביות. תיקיית קישור דינמי של 32 סיביות אינה יכולה ליצור מטלה נוספת, כאשר תוכנית של 16 סיביות קוראת לה.

הפונקציה CreateThread יוצרת את הידית של המטלה החדשה עם גישה מלאה למטלה זו. אם הקריאה ל-CreateThread אינה מספקת מתאר אבטחה, באפשרות התוכנית להשתמש בידית המוחזרת בכל פונקציה שדורשת ידית לאובייקט המטלה. כאשר התהליך מספק מתאר אבטחה, התוכנית מבצעת בדיקת גישה לכל השימושים העוקבים בידית, לפני שהיא מאפשרת גישה. אם בדיקת הגישה נדחית, התהליך המבקש אינו יכול להשתמש בידית כדי לגשת למטלה.

הפעלת המטלה מתחילה בפונקציה שמוגדרת על ידי הפרמטר lpStartAddress. כאשר פונקציה זו חוזרת, הפונקציה CreateThread משתמשת בערך DWORD המוחזר כדי לסיים את המטלה על ידי קריאה לפונקציה ExitThread. צריך להשתמש בפונקציה GetExitCodeThread כדי לקבל את הקוד המוחזר של המטלה, לאחר שהמטלה מסתיימת.

הפונקציה CreateThread עשויה להצליח, אפילו אם lpStartAddress מצביע לנתונים או קוד. אם כתובת ההתחלה אינה תקפה כאשר המטלה מופעלת, נוצרת הודעה על חריגה והמטלה מסתיימת. התוכנית מטפלת בסיום מטלה כתוצאה מכתובת התחלה שאינה תקפה, שנובעת משגיאה בעת יציאה של תהליך המטלה. התנהגות זו דומה לאסינכרוניות של CreateProcess, שבה הפונקציה יוצרת את התהליך, אפילו אם היא מתייחסת לתיקיות קישור דינמי (DLLs) שחסרות או שאינן תקפות.

הפונקציה CreateThread יוצרת את המטלה עם **עדיפות מטלה** (Thread Priority) של THREAD_PRIORITY_NORMAL. השתמש בפונקציות GetThreadPriority ו-SetThreadPriority כדי לקבל ולקבוע את ערך העדיפות של המטלה.

אובייקט המטלה נשאר במערכת עד שהמטלה מסתיימת והתוכנית סוגרת את כל הידיות שלה על ידי הקריאה ל-CloseHandle. התוכנית מפעילה באופן סדרתי את הפונקציות ExitProcess, ExitThread, CreateThread, CreateRemoteThread ותהליך שמתחיל (כתוצאה מקריאה ל-CreateProcess). רק אחד מתוך אירועים אלה יכול לקרות במרחב הכתובת בזמן נתון. כלומר, התהליך מקיים את המגבלות הבאות:

- ✰ בזמן התחלת התהליך ואתחול שגרות תיקיות הקישור הדינמי, התוכנית יכולה ליצור מטלות חדשות, אבל הן אינן מתחילות לפעול עד שהתהליך מסיים את אתחול תיקיות הקישור הדינמי.
- ✰ רק מטלה אחת בתהליך יכולה להיות בכל זמן נתון במהלך אתחול תיקיית קישור דינמי או בשגרה שאינה מוצמדת.
- ✰ הפונקציה ExitProcess אינה חוזרת, עד אשר אין עוד מטלות פעילות באתחול תיקיות הקישור הדינמי או בשגרות שאינן מוצמדות.

✍ **הערה:** מטלה שמשתמשת בפונקציות מתיקיות זמן הריצה של שפת C צריכה להשתמש בפונקציות זמן הריצה של C (run-time libraries) ו-beginthread (C runtime functions) ו-endthread לניהול המטלות, ולא להשתמש בפונקציות CreateThread ו-ExitThread. אם אין עושים זאת, עלול להיגרם אובדן זיכרון בזמן שהתוכנית קוראת ל-ExitThread.

כדי להבין יותר טוב את העיבוד שמבצעת הפונקציה CreateThread, התבונן בתוכנית **Simple_Thread**, שנמצאת בתקליטור המצורף לספר זה (בתיקיה Books\59285). התוכנית **Simple_Thread** יוצרת מטלה חדשה כל פעם שהמשתמש בוחר את האפשרות !Test מהתפריט, ומוציאה פלט לחלון שהמטלה התחילה. כאשר בוחרים באפשרות Exit, התוכנית משחררת כל מטלה בתור שלה, ומודיעה לך על הריסת המטלה. הקוד שבפונקציה WndProc יוצר את המטלה, כמו שרואים להלן:

```
case IDM_TEST:  // start up a thread.
   {
      DWORD dwChildId;
      CreateThread(NULL, 0, ChildThreadProc, hWnd,
                                           0, &dwChildId );
   }
   break;
```

16.11 הצגת אתחול המטלות

למדת שתוכניות יכולות ליצור מטלות כדי לבצע עיבודים נוספים במרחב ההפעלה שלהן. בכל פעם שיוצרים מטלה חדשה, Windows מבצעת מספר משימות יסודיות בעת אתחול המטלה, כדי לאפשר לה להתחיל בעיבוד הנדרש.

תחילה, Windows מקצה לכל מטלה מחסנית נפרדת, במרחב הזיכרון בן 4GB של התהליך. כאשר התוכניות משתמשות **במשתנים סטטיים וגלובאליים** (Static And Global Variables), מטלות רבות יכולות לגשת למשתנים אלה בו-זמנית, כאשר יש חשש לפגיעה בתכולת המשתנים. מכיון ש-Windows יוצרת את **המשתנים המקומיים** (Local Variables) ואת **המשתנים האוטומטיים** (Automatic Varialbles) במחסנית המטלה, הם יותר חסינים בפני פגיעה שעלולה להיות במשתנים הגלובליים בתוכניות של ריבוי מטלות. למדת שצריך תמיד לנסות ולהשתמש במשתנים מקומיים או אוטומטיים, ולא במשתנים גלובליים. כלל זה תקף באופן משמעותי יותר כאשר משתמשים במטלות.

בשלב השני, Windows מקצה לכל מטלה קבוצת **אוגרים** (Registers) של המעבד, שנקראים **הקשר המטלה** (Thread's Context) ומאחסנת אותם במבנה מסוג CONTEXT. באפשרות התוכניות לברר מהי תכולת מבנה CONTEXT בכל זמן, כדי לדעת את מצב אוגרי המעבד ששייכים למטלה. כאשר מערכת ההפעלה מתזמנת את המעבד עבור המטלה (כלומר, מקצה לה זמן), היא מאתחלת את האוגרים האלה בתוך הקשר המטלה הזו. האוגרים מכילים את **מצביע ההוראה** (Instruction Pointer), שמזהה את כתובת ההוראה הבאה של המטלה המעבד צריך לבצע) ואת **מצביע המחסנית** (Stack Pointer, שמזהה את הכתובת של מחסנית המטלה).

לאחר שהמטלה מסיימת את אתחול המחסנית וההקשר, ובהנחה שהיא אינה במצב מושהה (Suspended State), היא מתחילה להתבצע מהשורה הראשונה של הפונקציה שהוגדרה בעת יצירת המטלה.

16.12 שלבי יצירת מטלות על ידי מערכת ההפעלה

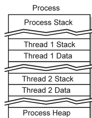
בסעיף קודם למדת שמערכת ההפעלה מבצעת מספר צעדים חשובים כאשר היא מקצה מטלות, שהתוכנית משתמשת בהן לפעולתה. למעשה, מערכת ההפעלה מבצעת ששה צעדים מוגדרים בכל פעם שהיא יוצרת מטלה חדשה, כפי שמפורטים להלן:

1. מקצה **אובייקט גרעין מטלה** (Thread Kernel Object), כדי לזהות ולנהל את המטלה החדשה שנוצרה. אובייקט הגרעין מחזיק הרבה ממידע המערכת כדי שיוכל לנהל את המטלה. ערך ידית אובייקט גרעין המטלה מוחזר על ידי הפונקציה CreateThread.
2. מאתחלת את ערך היציאה של המטלה ל-STILL_ACTIVE, וקובעת את **מונה ההשהיה** (Suspend Count) של המטלה ל-1 (שני הערכים מוחזקים על ידי Windows באובייקט גרעין המטלה).
3. מקצה מבנה CONTEXT עבור המטלה החדשה.
4. מכינה את מחסנית המטלה על ידי שמירת אזור במרחב הכתובת, משריינת עבור האזור שני דפים מאמצעי האחסון הפיסי, קובעת את ההגנה של אזור האחסנה ששוריין עבור PAGE_READWRITE, וקובעת לתכונה PAGE_GUARD את ערך הדף השני.
5. ממקמת את הערכים lpStartAddr ו-lpvThread בראש המחסנית, כך שהמטלה החדשה רואה אותם כפרמטרים שמועברים לפונקציה StartOfThread (רק אם הקוד משתמש בתיקיית זמן ריצה של מהדר C).
6. מאתחלת את אוגר המחסנית שבמבנה CONTEXT של המטלה, כדי שיצביע לערכים ש-Windows הציבה במחסנית בצעד 5. אחר כך, מערכת ההפעלה מאתחלת את אוגר **מצביע ההוראה** (Instruction Pointer) כדי להצביע לפונקציה הפנימית שהיא מפעילה, לפני הפעלת ההוראה הראשונה בפונקציית האתחול של המטלה.

16.13 קביעת גודל המחסנית של המטלה

בסעיף 16.10 למדת שבאפשרות התוכניות להגדיר את גודל מחסנית המטלה עבור הפונקציה CreateThread. חשוב לשים לב לכך. לאחר ש-Windows יוצרת את המטלה, התוכנית אינה יכולה לשנות את גודל המחסנית בצורה אמינה. במקום זאת, Windows מאפשרת למחסנית לצמוח בצורה דינמית ככל שיידרש.

אם אינך מציין גודל למחסנית המטלה, Windows מקצה מחסנית שגודלה כגודל המחסנית של המטלה הראשית. CreateThread יוצרת את המחסנית במרחב כתובת הזיכרון של התהליך. אפשר להציג את יצירת המחסנית בצורה ברורה במודל לוגי של מרחב המחסנית המוקצה, כמו שרואים בתרשים 16.5.



תרשים 16.5: מרחב המחסנית של התהליך, המטלה הראשית והמטלה המשנית.

כאשר Windows מקצה מרחב מחסנית עבור מטלות נוספות, היא עושה זאת בדרך כלל מתחת למרחב המחסנית של התהליך, ובמרחק של סגמנט אחד. Windows מקצה את מרחב המחסנית של המטלה בצורה וירטואלית, וכך היא יכולה להעביר את מחסנית המטלה בסביבת העבודה כשיש צורך בכך. לדוגמה, אם התהליך מאתחל את המטלה המשנית, ואחר כך יוצר מערך רב מימדי גדול מאוד (נניח, [1024,64] תווים), Windows חייבת להיות מסוגלת להעביר את המחסנית המשנית בסביבת העבודה, כדי להגן עליה מפני המטלה הראשית שעלולה לכתוב בצורה לא מכוונת בשטח המחסנית המשנית.

16.14 קבלת ידית אל המטלה הנוכחית או אל התהליך

ככל שהתוכניות הופכות להיות יותר ויותר מורכבות, ייתכנו מקרים בהם חייבות התוכניות לקבל בצורה דינמית ידית אל מטלה הנוכחית או אל התהליך הנוכחי. פעולה זו פשוטה למדי. באפשרות התוכניות לקרוא לפונקציה GetCurrentThread או GetCurrentProcess, כדי לקבל בכל זמן את הידית המדומה של המטלה הנוכחית או של התהליך הנוכחי, כפי שהם בתוכנית (ידית זו נקראת **ידית מדומה**, Pseudo-Handle, מכיון שהערך שלה הינו היחיד בעל משמעות במטלה הנוכחית או בתהליך הנוכחי). כותבים פונקציות אלו כמו בהגדרה שלהלן:

```
HANDLE GetCurrentThread(VOID);
HANDLE GetCurrentProcess(VOID);
```

צריך לשים לב לכך שהידיות המדומות ששתי הפונקציות מחזירות הן חסרות משמעות מחוץ לתהליך הנוכחי. כדי למסור את הידית של מטלה או של תהליך לתהליך אחר, חובה להשתמש בפונקציה DuplicateHandle.

כדי להבין יותר טוב את פעולת הפונקציות GetCurrentThread ו-GetCurrentProcess, התבונן בתוכנית **Show_Current**, שנמצאת בתקליטור המצורף לספר זה (בתיקיה Books\59285\Chap16b). התוכנית **Show_Current** יוצרת מטלות, זו אחר זו, ומציגה מידע אודות המטלות והתהליך.

16.15 ניהול זמן העיבוד של המטלה

כמו שניתן לדמיין, קביעת הזמן הדרוש לתהליך כדי לבצע פעילות מסוימת **בסביבה מרובת מטלות** (Multi-Threaded Environment) הרבה יותר קשה מאשר **בסביבה של מטלה אחת** (Single-Threaded Environment). לדוגמה, עשויה להיות בתהליך מטלה כלשהי שעורכת חישוב של אלגוריתם מורכב בזמן שמטלות שפועלות בתהליכים אחרים מתחרות בה לקבלת זמן המעבד. במקרה זה, ברור שתהליך החישוב יימשך זמן רב יותר מאשר במקרה שהיה פועל בעצמו, ללא תחרות. מכיון שתהליך זה יהיה בדרך כלל בעדיפות נמוכה, הדבר ישפיע על משך הביצוע הכולל שלו, שיהיה שונה מאוד מזמן המעבד נטו. מכאן שצריך להשתמש בשיטה שונה לרישום זמן העיבוד של המטלה (זמן מעבד) בסביבה מרובת מטלות, לעומת שיטת הרישום בסביבה של מטלה אחת. עשית זאת בוודאי בעת שהשתמשת בפונקציה clock לחישוב זמן העיבוד של התוכנית בסביבת העבודה DOS.

תוכניות מרובות מטלות חייבות להשתמש בפונקציה שבודקת את זמן הפעולה של כל מטלה בנפרד, במקום השיטה הפשוטה שהזכרנו. כדי לבצע עיבוד מסוג זה משתמשים ב-Windows בפונקציה GetThreadTimes. פונקציה זו מרכזת מידע על הזמן של המטלה שהוגדרה. משתמשים בפונקציה GetThreadTimes כמו בהגדרה זו:

```
BOOL GetThreadTimes(
  HANDLE hThread,         // specifies the thread of interset
  LPFILETIME lpCreationTime,  // when the thread was created
  LPFILETIME lpExitTime,      // when the thread was destroyed
  LPFILETIME lpKernelTime,    // time the thread has spent in
                              // kernel mode
  LPFILETIME lpUserTime       // time the thread has spent in
                              // user mode
);
```

הפונקציה GetThreadTimes מקבלת את הפרמטרים שמפורטים בטבלה 16.5.

טבלה 16.5: הפרמטרים שמקבלת הפונקציה **GetThreadTimes**.

פרמטר	תיאור
hThread	ידית פתוחה שמגדירה את המטלה שעבורה מרכזים את זמן העיבוד. עליך ליצור ידית זו עם קוד הגישה THREAD_QUERY_INFORMATION.
lpCreationTime	מצביע למבנה מסוג FILETIME שמקבל את זמן יצירת המטלה.
lpExitTime	מצביע למבנה מסוג FILETIME שמקבל את זמן היציאה מהמטלה. אם עדיין לא יצאנו מהמטלה, התכולה של מבנה זה אינה מוגדרת.

פרמטר	תיאור
lpKernelTime	מצביע למבנה מסוג FILETIME שמקבל את כמות הזמן שהמטלה הופעלה בו ב**מצב גרעין** (Kernel Mode).
lpUserTime	מצביע למבנה מסוג FILETIME שמקבל את כמות הזמן שהמטלה הופעלה בו ב**מצב משתמש** (User Mode).

כאשר הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, הערך המוחזר הוא אפס. הפונקציה GetThreadTimes משתמשת במבנה נתונים מסוג FILETIME להצגת כל סוגי הזמן. מבנים אלה מכילים שני ערכים בני 32 סיביות שמתחברים יחד כדי ליצור מונה בן 64 סיביות ביחידות זמן של 100 ננו-שניות (ננו-שנייה - חלק מיליארד של שנייה). זמן יצירת המטלה וזמן היציאה הם נקודות זמן שבאות על ידי ביטוי במבנה FILETIME כפרק הזמן שעבר מאז אמצע הלילה של הראשון לינואר משנת 1601, בגריניץ' אנגליה. ממשק התכנות Win32 API כולל מספר פונקציות שהיישום יכול להשתמש בהן כדי להפוך ערכים כאלה לתבניות זמן יותר כלליות, מובנות וגם שימושיות.

משכי הזמן של מצב גרעין המטלה ושל מצב המשתמש במטלה נמדדים בננו-שניות. לדוגמה, אם מטלה נמצאת למשך שנייה אחת במצב גרעין, הפונקציה GetThreadTimes ממלאת את מבנה FILETIME שגודלו 64 סיביות ומוגדר על ידי הפרמטר lpKernelTime בערך עשרה מיליון, שהינו מספר יחידות הזמן של 100 ננו-שנייה שיש בשנייה אחת.

16.16 ניהול זמן העיבוד במערכת של ריבוי מטלות

למדת בסעיף קודם שבאפשרות התוכניות להשתמש בפונקציה GetThreadTimes כדי לדעת את זמן הביצוע של כל מטלה. לפעמים נדרש בתוכניות מידע מפורט על זמן הביצוע של מטלות רבות בתהליך. לדוגמה, אנו זקוקים למידע זה כדי ללמוד על פעולת המערכת ולדעת אילו מטלות בתהליך איטיות, ואם יש מטלה אחת או יותר שגורמת להאטה בפעולת התהליך.

במקרים כאלה, התוכניות יכולות להשתמש בפונקציה GetProcessTimes כדי להשיג מידע זמן אודות התהליך המופעל. משתמשים בפונקציה GetProcessTimes בתוכניות, כמו שרואים בהגדרה שלהלן:

```
BOOL GetProcessTimes(
  HANDLE hProcess,          // specifies the process of interset
  LPFILETIME lpCreationTime, // when the process was created
  LPFILETIME lpExitTime,     // when the process was destroyed
  LPFILETIME lpKernelTime,   // time the process has spent in
                             // kernel mode
  LPFILETIME lpUserTime      // time the process has spent in
                             // user mode
);
```

הפונקציה GetProcessTimes מקבלת את הפרמטרים שמפורטים בטבלה 16.5.

כאשר הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, הערך המוחזר הוא אפס. הפונקציה GetProcessTimes משתמשת במבנה נתונים מסוג FILETIME להצגת כל סוגי הזמן. מבנים אלה מכילים שני ערכים בני 32 סיביות שמתחברים יחד כדי ליצור מונה בן 64 סיביות של 100 ננו-שניות יחידות זמן. זמן יצירת התהליך וזמן היציאה הם נקודות זמן שבאות על ידי ביטוי במבנה FILETIME ככמות הזמן שעבר מאז אמצע הלילה של ה-1 בינואר שנת 1601, בגריניץ' באנגליה. ממשק Win32 API כולל מספר פונקציות שהיישום יכול לנצל, כדי להפוך ערכים כאלה לתבניות יותר כלליות, מובנות ושימושיות.

הזמנים של מצב גרעין התהליך ומצב המשתמש הם משכי זמן שנמדדים בננו-שניות. לדוגמה, אם תהליך נמצא שנייה אחת במצב גרעין, הפונקציה GetProcessTimes ממלאת את המבנה FILETIME שמוגדר על ידי הפרמטר lpKernelTime בעל 64 הסיביות בערך של עשרה מיליון, שהוא מספר יחידות 100 ננו-שנייה שיש בשנייה אחת.

התקליטור המצורף לספר מכיל את התוכנית **Show_ThPr_Times** (בתיקיה Books\59285) שפותחת סדרת מטלות ומחזירה את זמן העיבוד של כל אחת מהן. היא מחזירה גם את זמן העיבוד של התוכנית כולה.

16.17 להבין טוב יותר את הפונקציה GetQueueStatus

ככל שעובדים עם מטלות, פוגשים במצבים שבהם מתרחשים אירועים (כמו הקשות במקשים, וכדומה), אשר מטופלים על ידי המטלה הראשית שנמצאת בפונקציה WndProc שמעבדת את הודעות התוכנית. עם זאת, כשיש מטלת בן מושהית, השליטה במידע ההודעות הופך להיות מורכב יותר. מכיון שהמטלה כבר מושהית (מסיבה כלשהי), פונקציית ההודעה של המטלה מחזיקה את ההודעות האלו עבור המטלה בתור ההודעות של המטלה. כדי לדעת את תכולת תור ההודעות לאחר שהמטלה המושהית מופעלת מחדש, התוכניות יכולה לנצל את שירותי הפונקציה GetQueueStatus. פונקציה זו מחזירה דגלים שמציינים את סוגי ההודעות שנמצאות בתור ההודעות של המטלה הקוראת לה. את הפונקציה GetQueueStatus כותבים בתוכנית כך:

```
DWORD GetQueueStatus(UINT flags);
```

הפרמטר flags מגדיר את דגלי המצב אשר מציינים את סוג ההודעות שהפונקציה צריכה לבדוק. פרמטר זה יכול להיות צירוף של הערכים שמפורטים בטבלה 16.6.

הערך המוחזר במילת הסדר הגבוהה מציין את סוגי ההודעות שנמצאות כרגע בתור. הערך של מילת הסדר הנמוך מציין את סוגי ההודעות ש-Windows הוסיפה לתור ועדיין נמצאות בו מאז הקריאה האחרונה לפונקציה GetQueueStatus, GetMessage, או לפונקציה PeekMessage.

נוכחות הדגל _QS בערך המוחזר אינה מבטיחה שקריאה עוקבת לפונקציה PeekMessage או GetMessage תחזיר הודעה. GetMessage ו-PeekMessage מבצעות סינון פנימי שיכול לגרום לתוכנית לטפל בהודעה באופן פנימי. מכיון שכך, עליך לחשוב על הערך המוחזר מ-GetQueueStatus רק כרמז לתוכנית לקרוא לאחת משתי הפונקציות, GetMessage או PeekMessage.

טבלה 16.6: הערכים האפשריים של הדגלים שהפרמטר **flags** יכול לקבל.

ערך	פירוש
QS_ALLEVENTS	קלט, WM_TIMER, WM_PAINT, WM_HOTKEY, או הודעה ששוגרה נמצאים בתור.
QS_ALLINPUT	יש הודעה כלשהי בתור.
QS_HOTKEY	הודעת WM_HOTKEY נמצאת בתור.
QS_INPUT	הודעת קלט נמצאת בתור.
QS_KEY	הודעת WM_KEYUP, WM_KEYDOWN, WM_SYSKEYUP, או WM_SYSKEYDOWN נמצאת בתור.
QS_MOUSE	הודעת WM_MOUSEMOVE או הודעת לחצן עכבר (WM_LBUTTONUP, WM_RBUTTONDOWN, וכדומה) נמצאת בתור.
QS_MOUSEBUTTON	הודעת לחצן עכבר (WM_LBUTTONUP, WM_RBUTTONDOWN, וכדומה) נמצאת בתור.
QS_MOUSEMOVE	הודעת WM_MOUSEMOVE נמצאת בתור.
QS_PAINT	הודעת WM_PAINT נמצאת בתור.
QS_POSTMESSAGE	הודעה ששוגרה (אחרת מאלו שכבר מנינו ברשימה) נמצאת בתור.
QS_SENDMESSAGE	הודעה שנשלחה על ידי מטלה אחרת או על ידי יישום אחר נמצאת בתור.
QS_TIMER	הודעת WM_TIMER נמצאת בתור.

16.18 עיבוד חריגים שלא טופלו - Handling Unhandled Exceptions

מערכת ההפעלה Win32 ממקמת **פונקציה ברמה עליונה לעיבוד חריגים** (Top-Level Exception Handler) בתחילת כל מטלה ותהליך. המטרה היא לוודא שתוכניות מגיבות בצורה נכונה לחריגים שאינם מטופלים (למעשה, תוכניות אלו נסגרות מבלי להשפיע באופן שלילי על תהליכים אחרים). לפעמים, ייתכן שתרצה ללכוד בשיגרה מיוחדת את כל החריגים שאינם מטופלים; שיגרה זו תשמור את התוכנית/התהליך של המשתמש **בקובץ שיקום** (Recovery File). הפונקציה SetUnhandledExceptionFilter מאפשרת ליישום לבטל את הפונקציה ברמה עליונה לעיבוד בחריגים ש- Win32 מיקמה אותה בראש כל מטלה או תהליך.

אחרי הקריאה לפונקציה SetUnhandledExceptionFilter, אם מתרחשת חריגה בתהליך ש-Windows אינה מנפה אותו משגיאות כרגע, והחריגה אינה מטופלת על ידי מסנן החריגים של Win32, אז מסנן זה קורא ל**פונקציית מסנן החריגה** (Exception Filter Function) שמוגדרת על ידי הפרמטר lpTopLevelExceptionFilter. את הפונקציה SetUnhandledExceptionFilter כותבים בתוכניות, כמו בהגדרה שלהלן:

```
LPTOP_LEVEL_EXCEPTION_FILTER SetUnhandledExceptionFilter(
   LPTOP_LEVEL_EXCEPTION_FILTER lpTopLevelExceptionFilter);
```

הפרמטר lpTopLevelExceptionFilter מספק את הכתובת של פונקציית מסנן החריגה ברמה עליונה ש-Windows קוראת לה בכל פעם שהפונקציה UnhandledExceptionFilter מקבלת שליטה, ו-Windows אינה מנפה את התהליך משגיאות. כדי להגדיר עיבוד בנוהל ברירת המחדל על ידי הפונקציה UnhandledExceptionFilter, ערך הפרמטר lpTopLevelExceptionFilter צריך להיות NULL.

התחביר של פונקציית המסנן מתאים לזה של UnhandledExceptionFilter. פונקציית המסנן מקבלת פרמטר אחד מסוג LPEXCEPTION_POINTERS ומחזירה ערך מסוג LONG. פונקציית המסנן צריכה להחזיר אחד מהערכים שמפורטים בטבלה 16.7.

הפונקציה SetUnhandledExceptionFilter מחזירה את הכתובת של מסנן החריגה הקודמת שקשור לפונקציה. אם הערך המוחזר הוא NULL, פירוש הדבר שאין מסנן חריגה נוכחי ברמה עליונה. השימוש ב-SetUnhandledExceptionFilter מחליף את מסנן החריגה ברמה העליונה עבור כל המטלות הקיימות ואלו שיווצרו אחר כך בתהליך הקורא. התוכנית מפעילה את פונקציית החריגה שמוגדרת על ידי הפרמטר lpTopLevelExceptionFilter בהקשר של המטלה שגרמה לשגיאה. מכיון שפונקציית החריגה מופעלת בתוך מטלה, יש אפשרות שתשפיע על פונקציות לעיבוד בחריגים והיכולת שלהן להתאושש מחריגים מסוימים, כמו מחסנית שאינה תקפה.

טבלה 16.7: הערכים המוחזרים האפשריים של פונקציית המסנן.

ערך	פירוש
EXCEPTION_EXECUTE_HANDLER	מוחזר מ-UnhandledExceptionFilter ומפעיל את הפונקציה המתאימה לעיבוד החריגה. ערך זה בדרך כלל גורם לסיום התהליך.
EXCEPTION_CONTINUE_EXECUTION	מוחזר מ-UnhandledExceptionFilter וממשיך את ההפעלה מהנקודה של החריגה. שים לב שפונקציית המסנן חופשית לשנות את מצב ההמשך על ידי שינוי מידע החריגה שמסופק על ידי הפרמטר LPEXCEPTION_POINTERS שלה.
EXCEPTION_CONTINUE_SEARCH	ממשיך בהפעלה הרגילה של UnhandledExceptionFilter ; כלומר, מקבל את משמעות הדגלים SetErrorMod, או קורא לתיבת ההודעה הנשלפת העוסקת בשגיאת היישום (Application Error).

כדי להבין היטב את מהלך העיבוד של הפונקציה SetUnhandledExceptionFilter, התבונן בתוכנית **Handle_Exception**, שבתקליטור המצורף לספר זה. תוכנית זו יוצרת מסנן לחריגות שאינן מטופלות, ואחר כך היא מציגה איך היא לוכדת חריגים מטופלים בפונקציות עיבוד בחריגים, וכיצד היא לוכדת באמצעות המסנן שהיא יצרה את החריגים שאינם מטופלים. הפונקציות YourUnhandledExceptionFilter ו-WndProc של התוכנית **Handle_Exception** מכילות את הקוד שמבצע את העיבוד בפועל.

16.19 סיום מטלות

ככל שהתוכניות הופכות להיות יותר מורכבות, ייתכנו מקרים שבהם מטלה נכשלת, ואינה יכולה לסיים את פעולתה בצורה נורמלית. למדת שהתוכניות חייבות במקרה זה להפסיק, או להפסיק זמנית, כל מטלה שנכשלה, כדי שמערכת ההפעלה תפסיק לתזמן אותה לביצוע ולא תקצה לה זמן מעבד. תוכנית שנכשלת ואינה יכולה לסיים את פעולתה בצורה תקינה, גם לא תיסגר. על כן, התוכנית חייבת להורות למערכת ההפעלה לסיים את המטלה במקומה. כדי לעשות זאת, צריך לשלב בתוכניות השונות את הפונקציה TerminateThread. את הפונקציה TerminateThread כותבים כמו בהגדרה שלהלן:

```
BOOL TerminateThread(
  HANDLE hThread,        // handle to the thread
  DWORD dwExitCode       // exit code for the thread
);
```

הפרמטר hThread מזהה את המטלה שצריך לסיים. תחת Windows NT, הידית חייבת להיות בעלת גישה THREAD_TERMINATE. הפרמטר dwExitCode מגדיר את קוד היציאה עבור המטלה. צריך להשתמש בפונקציה GetExitCodeThread לקבלת ערך קוד היציאה של המטלה. אם הפונקציה מצליחה, הערך המוחזר שונה מאפס. אם הפונקציה נכשלת, הערך המוחזר הוא אפס.

התוכניות משתמשות ב-TerminateThread כדי לגרום למטלה לצאת (Exit). כאשר משתמשים ב-TerminateThread כדי לגרום למטלה לצאת, למטלת המטרה אין כל סיכוי להפעיל קוד כלשהו של המשתמש, והתוכנית אינה מקצה מחדש את המחסנית ההתחלתית של המטלה. התוכנית אינה מודיעה לתיקיות הקישור הדינמי (DLLs) שמוצמדות למטלה שהמטלה מסתיימת.

הפונקציה TerminateThread היא פונקציה בעלת פוטנציאל סיכון, ולכן עליך להשתמש בה רק במקרים הקשים ביותר. עליך לקרוא ל-TerminateThread רק אם אתה יודע בדיוק מה מבצעת מטלת המטרה, ויש לך שליטה על כל הקוד שמטלת המטרה עשויה להפעיל בזמן הסיום. לדוגמה, TerminateThread יכולה לגרום לבעיות הבאות:

- אם מטלת המטרה בעלת קטע קוד קריטי, Windows אינה משחררת את הקטע הזה.
- אם מטלת המטרה מפעילה קריאות גרעין מסוימות בשעת הקריאה ל-TerminateThread מפסיקה אותה, מצב הגרעין שקשור לעיבוד המטלה עלול להיפגע.
- אם מטלת המטרה מנהלת את המצב הגלובלי של תיקיית קישור דינמי משותפת, Windows יכולה להרוס את התיקיה הזו וגם להשפיע על תיקיות קישור דינמי אחרות שבשימוש.

מטלה יכולה להגן על עצמה מפני TerminateThread על ידי שליטה בגישה לידיות שלה. לידית המטלה שמוחזרת על ידי הפונקציות CreateThread ו-CreateProcess יש לה גישה מסוג THREAD_TERMINATE, לכן כל תוכנית קוראת שמחזיקה את אחת הידיות האלו יכולה לסיים את המטלה. אם מטלת המטרה היתה המטלה האחרונה של התהליך בשעה שהתוכנית קראה לפונקציה TerminateThread, התוכנית מסיימת גם את התהליך של המטלה. המצב של אובייקט המטלה הופך להיות מסומן, והוא משחרר מטלות אחרות כלשהן שהמתינו למטלה שתסתיים. מצב סיום המטלה משתנה מ-STILL_ACTIVE לערך של הפרמטר dwExitCode.

סיום מטלה אינו מסיר בהכרח את אובייקט המטלה מהמערכת. Windows מוחקת אובייקט מטלה כאשר התוכנית סוגרת את הידית האחרונה של המטלה.

כדי להבין יותר טוב את העיבוד שמבצעת הפונקציה TerminateThread, התבונן בתוכנית **Manip_Threads**, שנמצאת בתקליטור המצורף לספר זה (בתיקיה Books\59285). תוכנית זו מאפשרת למשתמש ליצור מטלה, ואחר כך להשהות אותה (הקש על צירוף המקשים Alt+S), להפעיל את המטלה מחדש (הקש Alt+R) או לסיים את המטלה (הקש Alt+K). התוכנית נמנעת מלעסוק בעניינים הקשורים בסיום מטלה, בכך שהמטלה אינה מבצעת דבר, ורק מאפשרת למשתמש לגשת למטלה האחרונה שנוצרה. הפונקציות ThreadProc ו-WndProc של התוכנית **Manip_Threads** מכילות את הקוד שמבצע את העיבוד הנדרש.

16.20 קביעת זיהוי (ID) של מטלה או תהליך

למדת בסעיף 16.4 שפעמים רבות דרוש לתוכניות מזהה זמני, או ***ידית מדומה*** (Pseudo-Handle) אל המטלה הנוכחית או אל התהליך הנוכחי. לפעמים התוכניות דורשות ידית קבועה או ערך ייחודי אחר כדי לזהות מטלה או תהליך במערכת. ממשק Win32 API מספק שתי פונקציות, GetCurrentThreadId ו- GetCurrentProcessId, שמאפשרות לתוכניות להשיג ערך DWORD מיוחד שמערכת ההפעלה משתמשת בו כדי לזהות מטלה או תהליך בצורה פנימית. כותבים את הפונקציות האלו בתוכניות, כמו בהגדרה שלהלן:

```
DWORD GetCurrentThreadId (void);
DWORD GetCurrentProcessId (void);
```

הפונקציה GetCurrentThreadId מחזירה את מזהה המטלה (Tread Identifier) של המטלה הקוראת, שהוא למעשה הערך המוחזר של המטלה הקוראת. כל זמן שהמטלה פועלת ועד לסיומה, מזהה המטלה מאפשר זיהוי שלה במערכת באופן חד-ערכי.

באופן דומה, הפונקציה GetCurrentProcessId מחזירה את מזהה התהליך (Process Identifier) של התהליך הקורא. פונקציה זו אינה מקבלת פרמטרים. הערך המוחזר הוא מזהה התהליך של התהליך הקורא. כל עוד התהליך פועל ועד שהוא מסתיים, מזהה התהליך מאפשר את זהויו במערכת באופן חד-ערכי.

התוכנית **ShowCurrent** שהוצגה בסעיף 16.4 משתמשת בשתי הפונקציות GetCurrentProcessID ו-GetCurrentThreadID.

✍ **הערה:** בדומה לידית, המילה הכפולה DWORD שמוחזרת על ידי הפונקציות GetCurrentProcessID או GetCurrentThreadID מכילה ערך שמאפשר זיהוי חד-ערכי של המטלה או של התהליך על ידי מערכת ההפעלה. אל תבלבל בין הערכים Process ID או Thread ID לבין הידיות המדומות שמוחזרות על ידי GetCurrentProcess ו-GetCurrentThread.

16.21 תזמון מטלות על ידי מערכת ההפעלה

מערכת ההפעלה Win32 היא מערכת הפעלה **מרובת מטלות** (Multi-Threded). מערכת ההפעלה Win32 יכולה לטפל במספר גדול של מטלות עוקבות או תהליכים, הפועלים בהדדיות זה עם זה. כמו שהסברנו כבר, אנו רואים שמערכת ההפעלה מטפלת במספר מטלות יותר מהר, או בצורה אחרת מאשר באחרות. לדוגמה, מערכת ההפעלה Win32 נוטה לקבוע עדיפות גבוהה יותר למטלות שבתהליך הנוכחי, מאשר לאלו שבתהליכים המופעלים ברקע.

מערכת ההפעלה Win32 מכניסה כל מטלה שמבקשת זמן מעבד (כלומר, כל המטלות הפעילות) **לרשימה** (List), שהיא למעשה מבנה **תור** (Queue), על פי רמת העדיפות שיש לה. בסעיף 16.22 תמצא הסבר של **רמות העדיפות** (Priority Levels). כאשר המערכת מקצה זמן מעבד למטלה, היא מתייחסת לכל המטלות בעלות אותה עדיפות כשוות. במלים אחרות, המערכת מקצה זמן מעבד למטלה הראשונה שבתור הנושאת את רמת העדיפות 31, ואחרי שפרק הזמן של המטלה הזו מסתיים, המערכת מקצה זמן מעבד למטלה הבאה בתור, בעלת עדיפות 31. לכן, חשוב לשים לב לכך שאם יש לפחות מטלה אחת בעדיפות 31 הצורכת את זמן המעבד ברציפות, ללא הפסקה, היא אינה מאפשרת למטלות בעלות עדיפויות נמוכות יותר לפעול. תוכניתנים קוראים לתנאי עדיפות זה **הרעבה** (Starvation). הרעבה מתרחשת, כאשר מספר מטלות מקבלות הרבה מזמן המעבד, ואינן מאפשרות למטלה אחרת כלשהי לפעול.

כאשר המערכת משלימה את ביצוע כל המטלות בעלות העדיפות 31, היא מתחילה להקצות זמן מעבד למטלות בעלות עדיפות 30. כאשר המערכת מסיימת לבצע את כל המטלות בעלות עדיפות 30, היא מתחילה להקצות זמן למטלות בעלות עדיפות 29, וכך הלאה. על כן, יכול להיות שנראה כאילו המטלות בעלות עדיפות נמוכה לעולם אינן מופעלות במערכת זאת. כמו שיתברר לנו בהמשך, אפילו המטלות בעלות העדיפות הגבוהה ביותר אינן צורכות זמן מעבד ברציפות, וכך הן משחררות אותו לעיבוד של מטלות בעלות עדיפות נמוכה יותר.

לבסוף, צריך לדעת שכאשר מטלה בעלת עדיפות נמוכה מופעלת - ואפילו אם היא נמצאת באמצע פרק הזמן שהוקצה לה - והמערכת מחליטה שמטלה בעלת עדיפות גבוהה ממתינה להפעלה, המערכת מפסיקה מיד את הפעלת המטלה בעלת העדיפות הנמוכה ומתחילות בהפעלת המטלה בעלת העדיפות הגבוהה. המטלה בעלת העדיפות הגבוהה **תמיד** מקדימה את המטלה בעלת העדיפות הנמוכה, ללא תלות בפעולה של המטלה בעלת העדיפות הנמוכה, או באיזה מצב הפעלה היא נמצאת.

שים לב:

מיקרוסופט שומרת לעצמה את הזכות לשנות את האלגוריתם שמערכות הפעלה מבוססות Win32 משתמשות בו לניהול תור המטלות. במערכות ההפעלה Windows 9x, Windows NT 3.51, ו- Windows NT 4.0 משובצות גרסאות שונות של אלגוריתם לניהול תור המטלות. עליך להכיר את שיטת עדיפות המטלות

ואיך צריך להשתמש בה כראוי. עם זאת, אל תבנה את התוכניות שלך על פי שיטת ניהול עדיפויות כלשהי של מערכת הפעלה מסוימת, מכיון שמיקרוסופט עשויה לשנות את אלגוריתם ניהול תורי העדיפויות בגרסאות עתידיות של אותה מערכת הפעלה.

16.22 רמות עדיפות (Priority Levels)

בסעיף קודם למדת שמערכת ההפעלה Win32 מנהלת את תורי כל המטלות הפעילות ומתזמנת את הפעלתן, על פי **רמת העדיפות** (Priority Level) הנוכחית שיש להן. תחום רמות העדיפות הוא מ-0 (העדיפות הנמוכה ביותר) עד 31 (העדיפות הגבוהה ביותר). מערכת ההפעלה מקצה את רמת העדיפות "אפס" למטלת מערכת מיוחדת הקרויה **מטלת דף אפס** (Zero Page Thread). מטלת דף אפס אחראית לאיפוס דפים חופשיים כלשהם שבמערכת, כאשר אין מטלות כלשהן שחייבות לבצע עבודה כלשהי. אין מטלה אחרת כלשהי שיכולה להיות בעלת רמת עדיפות אפס, מלבד מטלה מיוחדת זו.

כאשר יוצרים מטלות, לא משתמשים במספרים כדי להקצות להן רמות עדיפות. במקום זאת, המערכת משתמשת בהליך בן שני צעדים כדי לקבוע את רמת העדיפות של המטלה. הצעד הראשון הוא להקצות **מחלקת עדיפות** (Priority Class) לתהליך. מחלקת העדיפות של התהליך אומרת למערכת איזו עדיפות התהליך מבקש בהשוואה לתהליכים מופעלים אחרים. בסעיף הבא תמצא הסבר נוסף אודות מחלקות עדיפות. הצעד השני הוא להקצות רמת עדיפות יחסית לכל מטלה ששייכת לתהליך.

כאשר יוצרים לראשונה מטלה בתהליך, רמת העדיפות שלה היא כמו זו של של התהליך עצמו. בסעיף 16.25 תלמד להשתמש ב- Win32 API כדי לשנות את רמת העדיפות היחסית של מטלה.

16.23 מחלקות העדיפות של Windows (Priority Classes)

כמו שהוסבר בסעיף קודם, Windows משתמשת בהליך בן שני צעדים כדי לקבוע את עדיפות המטלה. הצעד הראשון הוא לקבוע את מחלקת העדיפות של התהליך. Win32 תומכת בארבע מחלקות עדיפות שונות: **המתנה** (idle), **רגילה** (Normal), **גבוהה** (High), **זמן אמת** (Realtime). טבלה 16.8 מפרטת את מחלקות העדיפות.

טבלה 16.8: מחלקות העדיפות האפשריות.

עדיפות	פירוש
HIGH_PRIORITY_CLASS רמת עדיפות גבוהה	תהליך שמבצע משימות בזמן קריטי, שמערכת ההפעלה חייבת להפעילן מיד, כדי שהתהליך יפעל בצורה נכונה. מטלות תהליך בעל עדיפות

עדיפות	פירוש
HIGH_PRIORITY_CLASS רמת עדיפות גבוהה	מחלקה גבוהה קודמות למטלות של תהליכים בעלי מחלקת עדיפות רגילה או המתנה. דוגמה לכך היא **רשימת המשימות** (Task List) של Windows, שחייבת להגיב במהירות כאשר המשתמש קורא לה, ללא תלות בעומס של מערכת ההפעלה. צריך לנהוג בזהירות גדולה כאשר משתמשים במחלקת עדיפות גבוהה, מכיון שיישום בעל מחלקת עדיפות גבוהה יכול להשתמש כמעט בכל זמן המעבד ולא להרפות ממנו לטובת משימות אחרות. רמת העדיפות של תהליך בעל HIGH_PRIORITY_CLASS היא 13.
IDLE_PRIORITY_CLASS רמת עדיפות בהמתנה	תהליך שהמטלות שלו מופעלות רק כאשר המערכת נמצאת בהמתנה, ומטלות של תהליך אחר שמופעל במחלקת עדיפות יותר גבוהה הקדימו אותן. לדוגמה, שומר מסך. תהליכי בן יורשים את מחלקת עדיפות ההמתנה. רמת העדיפות של IDLE_PRIORITY_CLASS היא 6.
NORMAL_PRIORITY_CLASS רמת עדיפות רגילה	תהליך נורמלי, ללא דרישות מיוחדות כלשהן לתזמון. עדיפות תהליך בעל NORMAL_PRIORITY_CLASS היא 8.
REALTIME_PRIORITY_CLASS רמת עדיפות זמן-אמת	תהליך בעל העדיפות הגבוהה ביותר האפשרית. המטלות של תהליך בעל מחלקת עדיפות זמן אמת מקדימות את המטלות של כל שאר התהליכים, כולל תהליכי מערכת ההפעלה שמבצעים משימות חשובות. לדוגמה, תהליך זמן אמת אשר מופעל לאורך זמן ארוך מדי עלול לגרום לכך שמטמון הדיסק לא יתרוקן, או לגרום לכך שהעכבר לא יגיב. רמת העדיפות של REALTIME_PRIORITY_CLASS היא 24.

דבר שיכול לעזור לך בעניין זה הוא הבנה טובה של השפעת רמות המחלקות השונות. לדוגמה, צריך להשתמש ב-HIGH_PRIORITY_CLASS רק כאשר יש צורך בכך. התהליך הנפוץ ביותר שמשתמש ב-HIGH_PRIORITY_CLASS הוא **הסייר** של Windows (Windows Explorer). אפילו אם מרבית מטלות **שולחן העבודה** (Desktop) נרדמות (Sleeps) במהלך הפעלה רגילה, המשתמשים מצפים מיישום זה להגיב כאשר ניגשים אליו. לכן, Windows נותנת למטלות הסייר את העדיפות הגבוהה, אשר מקדימה כמעט כל מטלה אחרת כאשר המשתמש בוחר באפשרות כלשהי של שולחן העבודה. אם יוצרים תוכניות שמשתמשות גם כן ב-HIGH_PRIORITY_CLASS, יכול להיות ששולחן העבודה לא יגיב כנדרש, ואפילו יכול להיות ש-Windows תחסום אותו.

בדרך כלל, תוכניות מסוג יישומי בקרה תפעלנה ברמה IDLE_PRIORITY_CLASS. לדוגמה, כאשר תכתוב יישום שמציג באופן מחזורי את כמות הזיכרון החופשי שבמערכת. מכיון שלא תרצה שהיישום יפריע לביצוע משימות קריטיות אחרות התלויות בזמן, עליך לקבוע את מחלקת התהליך המחזורי ל-IDLE_PRIORITY_CLASS.

Windows מקצה באופן אוטומטי את NORMAL_PRIORITY_CLASS לכל תהליך שאינך מקצה לו במפורש ערך אחר, ורצוי שהתוכניות שלך תפעלנה בדרך כלל ברמה זו. שים לב לכך, שכאשר המשתמש מביא תהליך ל**קידמה** (Fourground), מערכת ההפעלה מעלה את העדיפות היחסית שלו, כדי לספק מהירות פעולה טובה יותר. לדוגמה, מערכת Windows 9x מוסיפה אחד למונה רמת העדיפות של תהליך הקידמה.

ככלל, צריך להימנע כמעט תמיד מהשימוש ב-REALTIME_PRIORITY_CLASS בתוכניות רגילות, מכיון שעדיפות זמן אמת היא עדיפות גבוהה מאוד - ולמעשה, היא אפילו גבוהה מרוב מטלות הניהול של מערכת ההפעלה. המטלות שבמערכת אשר שולטות בעכבר והמקלדת, בעבודת דיסק ברקע, ואפילו המלכודת Ctrl+Alt+Del - כולן מופעלות בעדיפות נמוכה יותר מעדיפות זמן אמת. לתוכניות אשר משתמשות בעדיפות זמן אמת יש פעמים רבות השפעות משמעותיות על יישומי המשתמש.

הערה: המונח **קידמה** (Foreground) מקובל במחשבים המסוגלים לבצע יותר ממשימה אחת בו-זמנית. הקידמה היא הסביבה שבה המשתמש עצמו פעיל ביישום, בשעה שמשימות אחרות, כגון הדפסת מסמך, העתקת קבצים או שידור תקשורת - פועלות **ברקע** (Background).

16.24 שינוי מחלקות העדיפות של התהליך

למדת בסעיף קודם שמערכת ההפעלה Win32 מקצה באופן אוטומטי את העדיפות הנורמלית לכל תהליך חדש. עם זאת, פעמים רבות עלינו לשנות בתוכניות את מחלקת העדיפות הנוכחית של התהליך. אפשר להשתמש בפונקציות GetPriorityClass ו-SetPriorityClass כדי לנהל את מחלקת העדיפות של התהליך. הפונקציה GetPriorityClass מחזירה את מחלקת העדיפות עבור התהליך המוגדר. את הפונקציה GetPriorityClass כותבים בתוכניות כמו בהגדרה שלהלן:

```
DWORD GetPriorityClass(HANDLE hProcess);
```

הידית hProcess מזהה את התהליך. תחת Windows NT, הידית hProcess חייבת להיות בעלת זכות גישה PROCESS_QUERY_INFORMATION. אם הפונקציה GetPriorityClass מצליחה, הערך המוחזר הוא מחלקת העדיפות של התהליך המוגדר; אם הפונקציה נכשלת, הערך המוחזר הוא אפס. מחלקת העדיפות של התהליך המוגדר יכולה לקבל את אחת מרמות העדיפות המפורטות בטבלה 16.8.

הפונקציה SetPriorityClass יכולה לקבוע את מחלקת העדיפות של התהליך המוגדר. מחלקת העדיפות, יחד עם ערך העדיפות של כל מטלת תהליך, קובעים את רמת העדיפות הבסיסית של כל מטלה. כותבים את פונקציה SetPriorityClass בתוכניות כמו בהגדרה שלהלן:

```
BOOL SetPriorityClass(
  HANDLE hProcess,            // handle to the process
  DWORD dwPriorityClass       // priority class value
);
```

כמו הפונקציה GetPriorityClass, גם הידית hProcess מזהה את התהליך. תחת Windows NT, הידית hProcess חייבת להיות בעלת זכות גישה PROCESS_SET_INFORMATION. הפרמטר dwPriorityClass מגדיר את מחלקת העדיפות של התהליך. הפרמטר dwPriorityClass יכול לקבל ערך כלשהו מאלה שמפורטים בטבלה 16.8. כאשר הפונקציה SetPriorityClass מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, הערך המוחזר הוא אפס.

לכל מטלה יש רמת עדיפות בסיסית, אשר נקבעת על ידי Windows בהתבסס על ערך העדיפות של המטלה ועל מחלקת העדיפות של תהליך המטלה. המערכת משתמשת ברמת העדיפות הבסיסית של כל המטלות שאפשר להפעיל, כדי לקבוע איזו מטלה מקבלת את הקצאת הזמן הבאה של המעבד. הפונקציה SetThreadPriority מאפשרת לקבוע את רמת העדיפות הבסיסית של מטלה באופן יחסי למחלקת עדיפות התהליך שהיא שייכת לו. סעיף 16.25 משתמש בפונקציה SetThreadPriority כדי לקבוע את רמת העדיפות של המטלה.

בתקליטור המצורף לספר זה תמצא את התוכנית **Get_Set_Priority** (בתיקיה Books\59285). תוכנית זו מאפשרת למשתמש לבחור את מחלקת העדיפות של התהליך. לאחר כל בחירה, התהליך יבצע פונקציה הצורכת זמן מעבד ויציג את תוצאות הפונקציה יחד עם מחלקת העדיפות המוחזרת על ידי מערכת ההפעלה.

16.25 קביעת העדיפות היחסית של התהליך

למדת ש-Windows קובעת את רמת עדיפות המטלה בהתבסס על מחלקת העדיפות של התהליך שהיא שייכת לו ועל רמת העדיפות של המטלה. בסעיף 16.24 למדת להשתמש בפונקציה SetPriorityClass כדי לשנות את מחלקת העדיפות של התהליך. כדי לשנות את רמת העדיפות של מטלות בתהליך, צריך להשתמש בפונקציה SetThreadPriority, אשר קובעת את ערך העדיפות של המטלה המוגדרת. ערך זה, יחד עם מחלקת העדיפות של התהליך שהמטלה שייכת לו, קובעים את רמת העדיפות הבסיסית של המטלה. משתמשים בפונקציה SetThreadPriority כפי שמוצג בהגדרה שלהלן:

```
BOOL SetThreadPriority(
  HANDLE hThread,        // handle to the thread
  int nPriority          // thread priority level
);
```

הפרמטר hThread מזהה את המטלה אשר הפונקציה הולכת לקבוע את ערך העדיפות שלה. תחת Windows NT, הידית חייבת להיות בעלת זכות גישה THREAD_SET_INFORMATION. הפרמטר nPriority מגדיר את ערך העדיפות של המטלה. פרמטר זה יכול לקבל את אחד הערכים שמפורטים בטבלה 16.9.

כאשר הפונקציה SetThreadPriority מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, הערך המוחזר הוא אפס. למדת שלכל מטלה יש רמת עדיפות בסיסית, ערך עדיפות של המטלה ומחלקת העדיפות שנקבעה לתהליך. המערכת משתמשת ברמת העדיפות הבסיסית של כל המטלות שאפשר להפעיל, כדי לקבוע איזו מטלה מקבלת את פרק הזמן הבא של המעבד. המערכת מנהלת את רשימת המטלות בשיטת round-robin בכל רמת עדיפות (המושג round-robin מכוון לכך שכל מטלה מטופלת בנפרד, ללא תלות במטלה אחרת ובלי מעורבותה או ידיעתה), ורק כאשר אין מטלות שאפשר להפעיל ברמה יותר גבוהה, המערכת מתחילה לנהל את רשימות המטלות בעלות רמת עדיפות הנמוכה יותר.

טבלה 16.9: ערכי רמת העדיפות של המטלה עבור הפרמטר **nPriority**.

עדיפות	פירוש
THREAD_PRIORITY_ABOVE_NORMAL	מציין רמה אחת מעל לעדיפות הנורמלית של מחלקת העדיפות.
THREAD_PRIORITY_BELOW_NORMAL	מציין רמה אחת מתחת לעדיפות הנורמלית של מחלקת העדיפות.
THREAD_PRIORITY_HIGHEST	מציין שתי רמות מעל לעדיפות הנורמלית של מחלקת העדיפות.
THREAD_PRIORITY_IDLE	מציין רמת עדיפות בסיסית 1 עבור התהליכים IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, או HIGH_PRIORITY_CLASS, ורמת עדיפות בסיסית 16 עבור תהליכי REALTIME_PRIORITY_CLASS.
THREAD_PRIORITY_LOWEST	מציין שתי רמות מתחת לעדיפות הנורמלית של מחלקת העדיפות.
THREAD_PRIORITY_NORMAL	מציין עדיפות נורמלית של מחלקת העדיפות.

עדיפות	פירוש
THREAD_PRIORITY_TIME_CRITICAL	מציין רמת עדיפות בסיסית 15 עבור התהליכים IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, או HIGH_PRIORITY_CLASS, ורמת עדיפות בסיסית 31 עבור תהליכי REALTIME_PRIORITY_CLASS.

הפונקציה SetThreadPriority מאפשרת לקבוע את רמת העדיפות הבסיסית של המטלה, יחסית למחלקת העדיפות של התהליך שהיא שייכת לו. לדוגמה, הגדרת THREAD_PRIORITY_HIGHEST בקריאה לפונקציה SetThreadPriority עבור מטלה ותהליך עם IDLE_PRIORITY_CLASS קובעת את רמת העדיפות הבסיסית של המטלה ל-6. עבור תהליכים עם IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS ו-HIGH_PRIORITY_CLASS, המערכת מעלה דינמית את רמת העדיפות הבסיסית של המטלה בשעה שמתרחשים אירועים שחשובים למטלה (כמו למשל, מטלה אחרת שעוברת למצב המתנה). תהליכי REALTIME_PRIORITY_CLASS אינם מקבלים כל העלאה דינמית (מכיון שהם כבר נמצאים ברמה הגבוהה ביותר). כל המטלות מתחילות ברמה THREAD_PRIORITY_NORMAL.

צריך להשתמש במחלקת עדיפות של התהליך כדי להבדיל בין יישומי זמן קריטי ואלה שיש להם דרישת תזמון נורמלית או מתחת לנורמלית. צריך להשתמש בערכי העדיפות של המטלה, כדי להבדיל בין העדיפויות היחסיות של משימות התהליך. לדוגמה, מטלה שמטפלת בקלט עבור חלון יכול להיות ברמת עדיפות יותר גבוהה מאשר מטלה שמבצעת חישובים אינטנסיביים וצורכת זמן מעבד רב.

כאשר מנהלים עדיפויות, צריך להיות זהירים מאוד, כדי להבטיח שמטלה בעלת עדיפות גבוהה לא תשתלט על כל זמן המעבד הפנוי. מטלה בעלת רמת עדיפות בסיסית מעל 11 מתנגשת עם הפעולה הרגילה של מערכת ההפעלה. השימוש ב-REALTIME_PRIORITY_CLASS שלא לצורך, או בחוסר תשומת לב, עלול לגרום לכך שמטמון הדיסק לא יתרוקן, שהעכבר לא יגיב, וכדומה.

16.26 קבלת רמת העדיפות הנוכחית של מטלה

בסעיף קודם למדת שבאפשרות התוכניות להשתמש בפונקציה SetThreadPriority כדי לשנות את רמת העדיפות הנוכחית של מטלה. פעמים רבות, דרוש מידע אודות רמת העדיפות הנוכחית של מטלה, בדרך כלל כצעד לפני הקריאה ל-SetThreadPriority לשינוי רמת העדיפות הקיימת. הפונקציה GetThreadPriority מחזירה את ערך העדיפות של המטלה המוגדרת. ערך זה, יחד עם מחלקת העדיפות של התהליך

שהמטלה שייכת לו, קובעים את רמת העדיפות הבסיסית של המטלה. את הפונקציה GetThreadPriority כותבים כמו בהגדרה שלהלן:

```
int GetThreadPriority(
  HANDLE hThread,              // handle to thread
);
```

כמו בפונקציה SetThreadPriority, גם כאן הפרמטר hThread מזהה את המטלה. כאשר הפונקציה מצליחה, היא מחזירה את רמת העדיפות של המטלה; אם הפונקציה נכשלת, היא מחזירה THREAD_PRIORITY_ERROR_RETURN. כדי לקבל מידע מפורט אודות השגיאה, צריך לקרוא ל-GetLastError. רמת העדיפות של המטלה היא אחד הערכים שמפורטים בטבלה 16.9.

זכור, לכל מטלה יש רמת עדיפות בסיסית שנקבעת על ידי מערכת ההפעלה לפי ערך העדיפות של המטלה ומחלקת העדיפות של התהליך שהיא שייכת לו. מערכת ההפעלה משתמשת ברמת העדיפות הבסיסית של כל המטלות שאפשר להפעיל כדי לקבוע איזו מטלה תקבל את הקצאת הזמן הבאה של המעבד. מערכת ההפעלה מנהלת את רשימות (או תורי) המטלות בשיטת round-robin בכל רמת עדיפות, כפי שהוסבר בסעיף קודם, ורק כאשר אין מטלות שאפשר להפעיל ברמה יותר גבוהה, מערכת ההפעלה פונה אל רשימות המטלות ברמה היותר נמוכה.

שים לב:

תחת Windows NT, הידית חייבת להיות בעלת גישה THREAD_QUERY_INFORMATION.

16.27 קבלת הקשר מטלה

למדת ש-Windows מאחסנת את המידע אודות המטלה בתוך מבנה CONTEXT של המטלה (Thread's CONTEXT). ככל שהתוכניות מטפלות במטלות, ייתכן שיידרש להן מידע על **הקשר** (Context) המטלה. הפונקציה GetThreadContext מקבלת את ההקשר של המטלה המוגדרת. כותבים פונקציה זו בתוכניות כמו בהגדרה שלהלן:

```
BOOL GetThreadContext(
  HANDLE hThread,          // handle of thread with context
  LPCONTEXT lpContext      // address of context structure
);
```

הפרמטר hThread מזהה ידית פתוחה של המטלה, שהפונקציה צריכה לקבל את ההקשר שלה. הפרמטר lpContext מצביע לכתובת של מבנה מסוג CONTEXT שמקבל את ההקשר המתאים של המטלה המוגדרת. ערך האיבר ContextFlags של מבנה זה מגדיר איזה חלקים מתוך הקשר המטלה צריך לקבל. המבנה CONTEXT הוא מבנה שתלוי בסוג המעבד/המחשב שבו התוכנית פועלת. כרגע, יש מבני CONTEXT מוגדרים עבור המעבדים Intel, MIPS, Alpha, ו-PowerPC.

צריך להשתמש בפונקציה GetThreadContext לקבלת ההקשר של המטלה הרצויה. הפונקציה מאפשרת לתוכניות לקבל הקשר לפי בחירתן, בהתבסס על ערך האיבר ContextFlags של המבנה CONTEXT. המטלה המטפלת בפרמטר hThread מניחה שהיא נקייה משגיאות, אבל הפונקציה יכולה לפעול גם כאשר היא טרם נוקתה משגיאות. אי אפשר לקבל הקשר תקף עבור מטלה שנמצאת בפעולה. חייבים להשתמש תחילה בפונקציה SuspendThread כדי להשהות את המטלה, ורק אז - לקרוא ל-GetThreadContext.

כדי להבין יותר טוב את משמעות הדברים, עיין בתוכנית **GetThreadContext** שנמצאת בתקליטור המצורף (בתיקיה Books\59285).

שים לב:

תחת Windows NT, הידית חייבת להיות בעלת גישה THREAD_GET_CONTEXT למטלה.

16.28 הפסקה זמנית והפעלה מחדש של מטלות

בסעיפים קודמים למדת שהתוכניות יכולות ליצור מטלות במצב מושהה (על ידי השימוש בדגל CREATE_SUSPENDED עם הפונקציה CreateProcess או CreateThread). כאשר יוצרים מטלה מושהית, המערכת יוצרת את אובייקט הגרעין המזהה את המטלה, יוצרת את מחסנית המטלה ומאתחלת את אוגרי המעבד עבור המטלה בהתאם למבנה CONTEXT. בנוסף, הפונקציה היוצרת נותנת לאובייקט המטלה **מונה השהייה** (Suspend Count) התחלתי שערכו 1; כלומר, המערכת לעולם לא תקצה זמן מעבד כדי להפעיל את המטלה. כדי לאפשר למטלה להתבצע, מטלה אחרת חייבת לקרוא לפונקציה ResumeThread ולמסור לה את ידית המטלה המושהית, כדי שתוריד את מונה ההשהייה. כאשר ResumeThread מורידה את מונה ההשהייה לאפס, התוכנית מחדשת את הפעלת המטלה. את הפונקציה ResumeThread כותבים בתוכנית כמו בהגדרה שלהלן:

```
DWORD ResumeThread(HANDLE hThread);
```

הפרמטר hThread מגדיר ידית למטלה שהתוכנית רוצה לחדש את הפעלתה. באפשרותך להשהות מטלה מספר פעמים; אך צריך לקרוא ל-ResumeThread כמספר הפעמים שהשהית את המטלה לפני שהמטלה מתחילה שוב להתבצע.

הפונקציה ResumeThread בודקת את מונה ההשהייה של המטלה שמדובר בה. אם מונה ההשהייה שווה לאפס, המטלה אינה מושהית כרגע; אחרת, הפונקציה ResumeThread מורידה את מונה השהייה. אם התוצאה היא אפס, התוכנית מתחילה בהפעלת המטלה מחדש. אם הערך המוחזר הוא אפס, פירוש הדבר שלא השהית את המטלה; כאשר הערך המוחזר הוא 1 - השהיית את המטלה, אך התוכנית הפעילה אותה מחדש. אם הערך המוחזר גדול מ-1, המטלה עדיין מושהית.

שים לב שבזמן דווח על אירועי תהליכי ניפוי שגיאות, מוקפאות כל המטלות שבתהליך הדיווח. מערכת ההפעלה מצפה תוכניות ניפוי השגיאות להשתמש בפונקציות SuspendThread ו-ResumeThread כדי להגביל את קבוצת המטלות שאפשר להפעיל בתהליך. אפשר להפעיל מטלה בודדת "צעד-אחר-צעד", על ידי השהיית כל המטלות האחרות בתהליך, מלבד המטלה שמפעילים עליה את דוח אירוע הניפוי. המטלות האחרות אינן משוחררות על ידי פעולת "המשך" אם הן מושהות.

שים לב:

תחת Windows NT, הידית חייבת להיות בעלת גישה THREAD_SUSPEND_RESUME למטלה.

16.29 סינכרון מטלות (Thread Synchronization)

בסעיפים קודמים למדת ש-Windows תומכת בריבוי מטלות. בסביבה אשר בה מטלה אחת או יותר יכולות להתבצע בו-זמנית, פעמים רבות חשוב לאפשר לתוכנית לסנכרן (וגם לתזמן כראוי) את פעילות המטלות השונות. מערכת הפעלה מבוססת Win32 מספקת מספר **אובייקטי סינכרון** (Synchronization Objects, או אובייקטי תזמון) שמאפשרים למטלות לסנכרן את הפעולות שלהן עם מטלות אחרות. בסעיף 16.30, תלמד בפירוט אודות אובייקטי הסינכרון.

בדרך כלל, מטלה מסנכרנת את עצמה עם מטלה אחרת על ידי כל שהיא "שמה את עצמה בתרדמה" ("Putting Itself To Sleep"). כאשר המטלה נרדמת, מערכת ההפעלה אינה מקצה עבורה זמן מעבד, ולכן המטלה מפסיקה לפעול. לפני שהמטלה מועברת למצב "תרדמה", היא מודיעה למערכת ההפעלה מהו "האירוע המיוחד" ("Special Event") - כמו הקשת מקש, לחיצת עכבר, או סיום אלגוריתם - אשר יגרום לכך שהמטלה תתחיל שוב לפעול.

מערכת ההפעלה, בתורה, ערה לדרישת המטלה ומשגיחה כדי לראות אם או מתי האירוע המיוחד מתרחש. כאשר האירוע מתרחש, מערכת ההפעלה מעוררת את המטלה והופכת אותה למטלה מן המניין שאפשר לבחור בה שוב ולהקצות לה זמן מעבד. לבסוף, המעבד משלב את המטלה וממשיך בהפעלתה; כלומר, המטלה מסונכרנת עכשיו ומתוזמנת לקבלת זמן מעבד, יחד עם שאר המטלות הפעילות.

16.30 הגדרת חמשת אובייקטי התיזמון העיקריים

למדת בסעיף קודם ש-Windows תומכת בסוגים שונים של אובייקטי סינכרון. מתוך אובייקטים אלה, חמשת האובייקטים השימושיים ביותר הם: critical sections, mutexes, semaphores, events ו-waitable timers. בסעיפים שיבואו בהמשך נדון

בהרחבה בכמה מסוגים אלה. חשוב להכיר את ההגדרות הפשוטות של כל סוג, כפי שאפשר ללמוד מטבלה 16.10.

טבלה 16.10: חמשת סוגי אובייקטי הסינכרון העיקריים.

סוג	שימוש ופעולות
Critical section קטע קריטי	**קטע קריטי** הוא קטע קוד קטן, אשר דורש גישה בלבדית (Exclusive) אל נתונים משותפים כלשהם, לפני שהוא מתחיל להתבצע. מבין כל אובייקטי התזמון, הקטע הקריטי הוא הפשוט ביותר לשימוש. אפשר להשתמש בקטעים קריטיים כדי לסנכרן מטלות בתהליך בודד כלשהו.
Mutexes מוטציות	**מוטציות** דומות לקטעים קריטיים. אך כאן השימוש הוא לסינכרון גישות אל נתונים דרך תהליכים רבים. בנוסף לכך, מוטציות הן **אובייקטי גרעין** (Kernel Object); כלומר, התוכניות יוצרות מוטציות באופן מעשי על ידי שימוש בפונקציית API, כמו למשל CreateMutex.
Semaphores סֵמָפור (אַתָת)	התוכניות משתמשות באובייקטי **סמפור** (Semaphore) כדי למנות משאבים. מטלה אחת יכולה להשתמש בסמפור כדי למנות את מספר המשאבים הזמינים וכדי להקצות משאבים. לדוגמה, אם יש למחשב שלוש יציאות טוריות, תוכל ליצור סמפור עם מונה משאבים שערכו שלוש. בכל פעם שמטלה ניגשת ליציאה טורית, מונה משאב הסמפור מופחת באחת, ובכל פעם שמטלה משחררת יציאה טורית, מונה משאב הסמפור מוגדל באחת. לכן, מטלות יכולות לקרוא לסמפור ולהמתין עד שהוא הופך לזמין, לפני שהן מנסות לגשת ליציאות הטוריות. השוני לעומת מוטציות וקטעים קריטיים, **המטלות אינן הבעלים של הסמפורים**.
Events אירועים	**אירועי אובייקטים** הם התבנית הפשוטה ביותר של סינכרון אובייקטים, והם שונים במקצת ממוטציות וסמפורים. תוכניות משתמשות במוטציות ובסמפורים כדי לשלוט בגישה לנתונים או למשאבים. הן גם משתמשות באירועים כדי לסמן השלמת פעולה. תוכניות משתמשות על פי רוב באירועים כדי להתחיל מטלה נוספת לאחר שהמטלה הראשונה סיימה חלק מסוים ממהלך העיבוד שלה.
Waitable timers קוצב זמן ממתין	**קוצב זמן ממתין** הוא אובייקט גרעין שמאותת לעצמו בצורה מחזורית, בזמן מוגדר, או במרווחי זמן קבועים. אפשר לחשוב על waitable time כשעון אזעקה פנימי עבור התוכניות. לדוגמה, אולי תכתוב תוכנית לשילוב זמנים שמתריעה המשתמש בכל שעה על פגישות חדשות של אותה שעה. במקום להפעיל לולאה

סוג	שימוש ופעולות
Waitable timers קוצב זמן ממתין	באופן קבוע ולהמתין לשעה שתשתנה, באפשרות התוכנית ליצור קוצב זמן ממתין אשר מסמן לתוכנית שהזמן השתנה כל שעה. קוצבי זמן ממתינים קיימים רק בגירסאות של Windows NT 4 ומעלה. Windows 9x אינה תומכת בקוצבי זמן ממתינים.

16.31 יצירת קטע קריטי

למדת שהסוג הפשוט ביותר של סינכרון מטלות הוא באמצעות איתות של ״קטע קריטי״. קטע קריטי מאפשר לתוכניות לשלוט בגישה אל קטע מהנתונים או אל פונקציה כלשהי בתוכנית. התנאי הוא שרק מטלה אחת תיגש לנתונים אלה בכל זמן נתון, או שכל שאר המטלות הפנימיות בתהליך כבר סיימו את העיבוד שלהן לפני שנוהל ״קטע קריטי״ מתחיל להתבצע.

יצירת קטע קריטי קלה ופשוטה. קודם כל, על התוכנית להקצות בתהליך המופעל מבנה נתונים מסוג CRITICAL_SECTION. התוכנית חייבת להקצות את מבנה הנתונים הזה בצורה גלובלית, כך שמטלות שונות שבתהליך הנוכחי תוכלנה לגשת למופע CRITICAL_SECTION שנוצר בתוכנית. כלומר, מופע CRITICAL_SECTION צריך להיות משתנה גלובלי.

לאחר שהתוכנית מקצה את מבנה הנתונים מסוג CRITICAL_SECTION, היא חייבת לבצע שתי פעולות כדי ליצור את הקטע הקריטי ולהיכנס אליו. התוכנית חייבת לקרוא תחילה לפונקציה InitializeCriticalSection כדי לאתחל את הקטע הקריטי, ועליה לקרוא לפונקציה EnterCriticalSection כשהיא מוכנה להיכנס אליו. הפונקציה EnterCriticalSection ממתינה למטלה עד שתהיה לה בעלות על אובייקט הקטע הקריטי המוגדר. הפונקציה חוזרת, כאשר מערכת ההפעלה מעבירה למטלה הקוראת את הבעלות. את הפונקציה EnterCriticalSection כותבים בתוכנית כמו בהגדרה שלהלן:

```
void EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

הפרמטר lpCriticalSection מצביע אל אובייקט הקטע הקריטי. כדי לאפשר גישה בלבדית אל המשאב המשותף, כל מטלה קוראת לפונקציה EnterCriticalSection או TryEnterCriticalSection כדי לבקש בעלות על הקטע הקריטי, לפני שהיא מבצעת קטע קוד כלשהו, אשר ניגש אל המשאב המוגן. ההבדל הוא בכך שהפונקציה TryEnterCriticalSection חוזרת מיד, ללא קשר אם הצליחה להשיג בעלות על הקטע הקריטי או לא, בזמן שהפונקציה EnterCriticalSection חוסמת את המשך העיבוד עד אשר המטלה יכולה לקבל בעלות על הקטע הקריטי המבוקש. כאשר המטלה מסיימת להפעיל את קטע הקוד המוגן, היא משתמשת בפונקציה LeaveCriticalSection כדי לשחרר את הבעלות, ולאפשר למטלה אחרת לקבל את הבעלות ולגשת למשאב המוגן. המטלה השולטת על הקטע חייבת לקרוא לפונקציה LeaveCriticalSection בכל פעם שהיא נכנסת לקטע הקריטי. המטלה יכולה להיכנס אל הקטע הקריטי רק כאשר הפונקציות EnterCriticalSection או TryEnterCriticalSection מצליחות.

לאחר שמטלה מקבלת בעלות על קטע קריטי, היא יכולה לקרוא שוב לפונקציה EnterCriticalSection או לפונקציה TryEnterCriticalSection מבלי שההפעלה שלה תחסם. הדבר מונע מהמטלה לחסום את עצמה (כלומר, להפסיק את פעולתה) בזמן שהיא ממתינה לקטע קריטי שיש לה כבר בעלות עליו.

כל מטלה של תהליך יכולה להשתמש בפונקציה DeleteCriticalSection כדי לשחרר את משאבי המערכת שהוקצו על ידי התוכנית, אשר אתחלה את אובייקט הקטע הקריטי. לאחר שהמטלה קוראת לפונקציה DeleteCriticalSection, התוכנית אינה יכולה להשתמש יותר באובייקט הקטע הקריטי לסינכרון.

✍ **הערה:** למרות שהאיברים של מבנה הנתונים CRITICAL_SECTION מוגדרים בקובץ הכותר **winbase.h**, על התוכניות להימנע מלגשת לאיברי המבנה, מכיון ש-Windows מנהלת את המידע שנמצא בהם באופן פנימי, ושינוי האיברים יכול לגרום **לשגיאות מערכת פטליות**.

✍ **הערה:** **שגיאה פטלית** (Fatal Error) היא כל שגיאת עיבוד, שהתוכנית איננה יכולה להתאושש ממנה. לעיתים אפשר לצאת מהתוכנית מבלי להפעיל את המחשב מחדש, אבל סביר להניח שנתונים שלא אוחסנו ילכו לאיבוד.

16.32 קטע קריטי פשוט

למדת שיצירה ושימוש בקטע קריטי היא הליך בן שלושה צעדים. חייבים ליצור את הקטע, לאתחל אותו ולהיכנס אליו. לפני שתוכל לסנכרן מטלות על פי קטע קריטי, עליך לאתחל את הקטע הקריטי ולמסור את כתובת מבנה הנתונים CRITICAL_SECTION כפרמטר יחיד. כאשר מגיעים להתחלת הקטע הקריטי, התוכנית חייבת לקרוא לאחת משתי הפונקציות: EnterCriticalSection או TryEnterCriticalSection, ולהעביר שוב את הכתובת של מבנה הנתונים CRITICAL_SECTION כפרמטר יחיד. לאחר שהמטלה כבר מסונכרנת, הקטע הקריטי נשאר עד שהתוכנית עוזבת אותו, או עד שהיא מוחקת אותו.

כדי להבין יותר טוב את פעולות התוכנית בעת ניהול קטעים קריטיים, התבונן בתוכנית **Crit_Section**, שנמצאת בתקליטור המצורף לספר זה (בתיקיה Books\59285). התוכנית משתמשת בקטע קריטי, שנמצאת בו הוראה ל"הרדם" למשך חמש שניות, כדי לאפשר למטלה אחת להפעיל קטע קוד קריטי בכל רגע נתון. כאשר המשתמש בוחר באפשרות !Test, התוכנית יוצרת מטלה אחרת, אשר גם היא בתורה ממתינה לגישה שלה לקטע הקריטי. הקוד הפעיל של התוכנית **Crit_Section** נמצא בפונקציות ChildThreadProc ו-WndProc.

16.33 WaitForSingleObject - לסינכרון של שתי מטלות

פעילויות סינכרון רבות ממתינות למטלה אחת או יותר לפני שהן ממשיכות בעיבוד המטלה הנוכחית. כאשר המטלה הנוכחית מחכה למעבד שיחזור מפעילות במטלה אחרת, התוכניות צריכות להשתמש בפונקציה WaitForSingleObject, אשר חוזרת כאשר אחד האירועים הבאים קורה:

- ✰האובייקט המוגדר נמצא **במצב מסומן** (Signaled State).
- ✰פרק **פסק הזמן** (Time-Out) מסתיים.

את הפונקציה WaitForSingleObject כותבים בתוכנית, כמו בהגדרה שלהלן:

```
DWORD WaitForSingleObject(
  HANDLE hHndle,         // handle of object to wait for
  DWORD dwMilliseconds   // time-out interval in milliseconds
);
```

הפונקציה WaitForSingleObject מקבלת שני פרמטרים, hHandle ו-dwMilliseconds. הפרמטר hHandle הינו זיהוי האובייקט אשר הפונקציה צריכה לחכות לו. הפרמטר dwMilliseconds מגדיר את פסק הזמן במילישניות. הפונקציה WaitForSingleObject חוזרת כאשר פרק הזמן מסתיים, אפילו אם מצב האובייקט **אינו מסומן** (Non Signaled). כאשר dwMilliseconds שווה לאפס, הפונקציה בודקת את מצב האובייקט וחוזרת מיד. כאשר dwMilliseconds שווה ל-INFINITE, פסק הזמן של הפונקציה אינו מסתיים. טבלה 16.11 מכילה את רשימת סוגי האובייקטים שאפשר להגדיר את הידיות שלהם (כלומר, סוגי אובייקטים שהפונקציה WaitForSingleObject יכולה לחכות להם).

טבלה 16.11: האובייקטים אשר הפונקציה **WaitForSingleObject** יכולה לחכות להם.

סוג האובייקט	תיאור
Change notification הודעת שינוי מצב	הפונקציה FindFirstChangeNotification מחזירה את הידית. המצב של אובייקט **שינוי הודעת מצב** (Change Notification) הוא מסומן, כאשר מתרחש סוג שינוי מוגדר בתיקיה או בעץ תיקיות שמגדירים.
Console input קלט הקונסול	הפונקציות CreateFile או GetStdHandle מחזירות את הידית כאשר מגדירים את הערך CONIN$. מצב האובייקטים מסוג **קלט קונסול** (Console Input) הוא **מסומן** (Signaled), כאשר יש קלט שלא נקרא במאגר הקלט של הקונסול; והוא **אינו מסומן** (Non-Signaled) כאשר מאגר הקלט ריק.

סוג האובייקט	תיאור
Event אירוע	הפונקציות CreateEvent או OpenEvent מחזירות את הידית. שתי הפונקציות SetEvent ו-PulseEvent קובעות את מצב אובייקט ה**אירוע** (Event) למסומן בצורה ברורה. התוכניות חייבות להשתמש בפונקציה ResetEvent כדי **לאפס** (Reset) כל אובייקט **אירוע שמאופס ידנית** (Manual-Reset Event) ולהסב אותו למצב אינו מסומן. עבור אובייקט **אירוע שמאופס אוטומטית** (Auto-Reset Event), הפונקציה הממתינה מאפסת את מצב האובייקט לאינו מסומן לפני שהיא חוזרת. אפשר להשתמש באובייקטי **אירוע** גם עבור בפעולות חופפות, שבהם המערכת קובעת את המצב.
Mutex מוטציה	הפונקציות CreateMutex או OpenMutex מחזירות את הידית. מצב אובייקט מוטציה הוא מסומן כאשר אין למטלה כלשהי בעלות עליו. הפונקציה הממתינה דורשת בעלות על המוטציה עבור המטלה הקוראת, ומשנה את מצב המוטציה לאינו מסומן, כאשר מערכת ההפעלה מוסרת בעלות על המוטציה.
Process תהליך	הפונקציות CreateProcess או OpenProcess מחזירות את הידית. מצב אובייקט **התהליך** הוא מסומן, כאשר התהליך מסתיים.
Semaphore סמפור (אתת)	הפונקציות CreateSemaphore או OpenSemaphore מחזירות את הידית. אובייקט **סמפור** (Semaphore) מחזיק מונה שערכו בין אפס לבין ערך מקסימלי כלשהו. המצב שלו מסומן כאשר המונה שלו גדול מאפס - ואינו מסומן, כאשר המונה שלו שווה לאפס. אם המצב הנוכחי הוא מסומן, הפונקציה הממתינה מפחיתה את המונה באחד.
Thread מטלה	הפונקציות CreateProcess, CreateThread, או CreateRemoteThread מחזירות את הידית. מצב אובייקט **המטלה** מסומן כאשר המטלה מסתיימת.
Timer קוצב זמן	הפונקציות CreateWaitableTimer או OpenWaitableTimer מחזירות את הידית. הפעלת **קוצב הזמן** נעשית על ידי הקריאה לפונקציה SetWaitableTimer. קוצב זמן פעיל הוא במצב מסומן, כאשר ערכו מגיע לערך שהוגדר מראש. אפשר לבטל את פעולת הקוצב על ידי קריאה לפונקציה CancelWaitableTimer.

✍ **הערה:** ב- Windows NT, הידית חייבת להיות בעלת זכות גישה SYNCRONIZE.

אם הפונקציה WaitForSingleObject נכשלת, הערך המוחזר הוא WAIT_FAILED ; אם היא מצליחה, הערך המוחזר מציין את האירוע שגרם לה לחזור. הערך המוחזר עבור קריאה מוצלחת לפונקציה הוא אחד הערכים שמפורטים בטבלה 16.12.

טבלה 16.12: ערכי ההצלחה האפשריים המוחזרים על ידי הפונקציה WaitForSingleObject.

ערך	פירוש
WAIT_ABANDONED	האובייקט המוגדר הוא אובייקט מוטציה, אשר המטלה שיש לה בעלות על אובייקט המוטציה לא שיחררה אותו לפני שהסתיימה. מערכת ההפעלה מוסרת את הבעלות על אובייקט המוטציה למטלה הקוראת, וקובעת את המוטציה למצב "אינה מסומנת".
WAIT_OBJECT_0	מצב האובייקט המוגדר הוא "מסומן".
WAIT_TIMEOUT	פרק פסק הזמן הסתיים, ומצב האובייקט "אינו מסומן".

הפונקציה WaitForSingleObject בודקת את המצב הנוכחי של האובייקט המוגדר. אם מצב האובייקט אינו מסומן, המטלה הקוראת נכנסת למצב המתנה. המטלה מנצלת זמן עיבוד קטן מאוד בזמן שהיא ממתינה שמצב האובייקט יהפוך למסומן, או שפסק הזמן יסתיים. לפני ההפעלה, **פונקציית המתנה** (Wait Function) משנה את המצב של מספר סוגי אובייקטי סינכרון. השינוי מתרחש רק עבור האובייקט או האובייקטים אשר המצב המסומן שלהם גרם לפונקציה לחזור. לדוגמה, פונקציית ההמתנה מפחיתה את המונה של אובייקט סמפור באחד.

צריך להיות זהירים בעת שמשתמשים בפונקציית ההמתנה ובחילופי נתונים דינמיים. כאשר מטלה יוצרת חלונות כלשהם, היא חייבת לטפל בהודעות. חילופי נתונים דינמיים שולחים הודעות לכל החלונות שבמערכת. אם יש מטלה אשר משתמשת בפונקציית המתנה בלי פסק זמן מוגדר, המערכת ננעלת. לכן, אם יש מטלה שיוצרת חלונות, השתמש בפונקציה MsgWaitForMultipleObjects או בפונקציה MsgWaitForMultipleObjectsEx, ולא בפונקציה WaitForSingleObject.

כדי להבין יותר טוב את פעולת הפונקציה WaitForSingleObject, התבונן בתוכנית **Wait_Events**, שבתקליטור המצורף לספר זה (בתיקיה Books\59285). בכל פעם שהמשתמש בוחר באפשרות !Test מהתפריט, התוכנית מתחילה להפעיל מטלה שממתינה לאירוע, נרדמת, ואחר כך משחררת את האירוע. השימוש **באירועים מאופסים אוטומטית** (Auto-Reset Events), שמשנים את דגלי הסימון שלהם בצורה אוטומטית), מחייב את הגישה למטלה **במיון סדרתי** (Serial Order), מכיון שהאירועים מאפסים את הערכים שלהם בצורה אוטומטית למצב "אינו מסומן", כאשר המטלה הראשונה הממתינה מקבלת את האובייקט. קטעי העיבוד של התוכנית **Wait_Events** הרלוונטיים לענייננו נמצאים בפונקציות ChildThreadProc ו-WndProc.

WaitForMultipleObjects *16.34* - סינכרון מטלות רבות

תוכניות יכולות להשתמש בפונקציה WaitForSingleObject כדי לסנכרן מטלה עם אירוע יחיד שמאופס אוטומטית. אך השימוש הנפוץ יותר הוא, שהתוכניות דורשות שהעיבוד יימשך רק כאשר יש מופע של קבוצה אובייקטים אחת מסוימת, או יותר. במקרים כאלה, התוכניות יכולות להשתמש בפונקציה WaitForMultipleObjects. הפונקציה WaitForMultipleObjects חוזרת כאשר מתרחש אחד מהאירועים הבאים:

- ✰ אובייקט אחד כלשהו שמוגדר, או כל האובייקטים המוגדרים, נמצאים במצב מסומן (Signaled), או במצב איתות.
- ✰ משך פסק הזמן הסתיים.

משתמשים בפונקציה WaitForMultipleObjects כמו בהגדרה שלהלן:

```
DWORD WaitForMultipleObjects(
  DWORD nCount,              // number of handles in the
                             // object handle array
  CONST HANDLE *lpHandles,   // pointer to the
                             // object-handle array
  BOOL bWaitAll,             // wait flag
  DWORD dwMilliseconds       // time-out interval in
                             // Milliseconds
);
```

אחרי הקריאה, WaitForMultipleObjects חוזרת בגלל כישלון, הצלחה, או פסק זמן. אם הפונקציה WaitForMultipleObjects נכשלת, הערך המוחזר הוא WAIT_FAILED; ואם הפונקציה WaitForMultipleObjects מצליחה, הערך המוחזר מציין את האירוע אשר גרם לה לחזור. הערך המוחזר בעת הצלחה הוא אחד הערכים שמפורטים בטבלה 16.12.

הפונקציה WaitForMultipleObjects מחליטה אם אובייקט אחד או יותר מאלה שהמטלה ממתינה להם, קיים את קריטריון ההמתנה. אם אף לא אחד מהאובייקטים אשר המטלה ממתינה להם מקיים את קריטריון ההמתנה, המטלה הקוראת נכנסת למצב המתנה יעיל, שבו היא מנצלת זמן עיבוד קטן בשעה שהיא ממתינה לאובייקט אחד או יותר מאלה שהמטלה ממתינה להם, כדי שיקיימו את קריטריון ההמתנה. כאשר הפרמטר bWaitAll הוא אמת (True), פעולת ההמתנה של הפונקציה מסתיימת, רק כאשר מצב כל האובייקטים הופך למסומן. הפרמטר bWaitAll אינו משנה את מצב האובייקטים המוגדרים, עד שמצבי כל האובייקטים הוא מסומן. לדוגמה, מוטציה יכולה להיות מסומנת, אבל המטלה אינה מקבלת בעלות עליה עד שמצב שאר האובייקטים גם הוא מסומן. במשך זמן זה, יכול להיות שמטלה אחרת תקבל את הבעלות על המוטציה, ובכך היא קובעת את המצב שלה לאינו-מסומן.

טבלה 16.13: הפרמטרים של הפונקציה WaitForMultipleObjects.

פרמטר	תיאור
nCount	מגדיר את מספר ידיות האובייקטים שבמערך, אשר lpHandles מצביע עליו. מספר ידיות האובייקטים המקסימלי הוא MAXIMUM_WAIT_OBJECTS (קבוע שמוגדר במערכת, ואשר משתנה בכל התקנה).
lpHandles	מצביע למערך ידיות של אובייקטים. בטבלה 16.11 נמצאת רשימת סוגי האובייקטים אשר אפשר להגדיר את הידיות שלהן. המערך ש-lpHandles מצביע עליו, יכול להכיל ידיות של אובייקטים מסוגים שונים.
✍ **הערה:**	תחת Windows NT, הידיות חייבות להיות בעלות זכות גישה SYNCHRONIZE.
bWaitAll	מגדיר את סוג ההמתנה: אם **אמת** (True), הפונקציה חוזרת כאשר המצב של כל האובייקטים שבמערך אשר lpHandles מצביע עליו מסומנים; אם **שקר** (False), הפונקציה חוזרת כאשר המצב של אובייקט אחד כלשהו נקבע למסומן, ואז הערך המוחזר מציין את האובייקט שהמצב שלו גרם לפונקציה לחזור.
dwMilliseconds	מגדיר את פרק פסק הזמן במילישניות. הפונקציה חוזרת אם פרק הזמן מסתיים, אפילו אם התנאים שמוגרים על ידי הפרמטר bWaitAll אינם מתקיימים. אם dwMilliseconds שווה לאפס, הפונקציה בודקת את מצבי האובייקטים המוגדרים וחוזרת מיד. אם dwMilliseconds שווה ל-INFINITE (אינסוף), פרק פסק הזמן של הפונקציה לעולם אינו מסתיים.

לפני החזרה, פונקציית המתנה משנה את המצב של חלק מסוגי אובייקטי הסינכרון. השינוי מתרחש רק עבור האובייקט או האובייקטים, אשר מצב הסימון שלהם גרם לפונקציה לחזור. לדוגמה, מספר מטלות אחרות או תהליכים מקטינים את ערך המונה של אובייקט **הסמפור** באחד. הפונקציה WaitForMultipleObjects יכולה להגדיר במערך שמוצבע על ידי הפרמטר lpHandles ידית אחת או יותר מסוגי האובייקטים שנמצאים ברשימה שבטבלה 16.11.

כמו במקרה של הפונקציה WaitForSingleObjects, גם כאן חייבים להיות זהירים כאשר משתמשים בפונקציה WaitForMultipleObjects ובחילופי נתונים דינמיים (Dynamic-Data Exchange). כאשר מטלה יוצרת חלונות כלשהם, היא חייבת לטפל בהודעות שהם מפיקים. חילופי נתונים דינמיים שולחים הודעות לכל החלונות שבמערכת. המערכת ננעלת, כאשר יש לך מטלה אשר משתמשת בפונקציית המתנה ללא פונקציית פסק זמן. לכן, אם יש לך מטלה אשר יוצרת חלונות, השתמש בפונקציה MsgWaitForMultipleObjects או בפונקציה MsgWaitForMultipleObjectsEx, ולא בפונקציה WaitForSingleObject.

16.35 יצירת מוטציה (Creating A Mutex)

למדת שמוטציות (Mutexes) דומות לקטעים קריטיים. פרט לכך, באפשרות התוכניות להשתמש ב**מוטציות** כדי לסנכרן גישה לנתונים דרך אובייקטים רבים, במקום גישה דרך אובייקט אחד. כדי להשתמש במוטציה, התוכניות חייבות ליצור תחילה את המוטציה באמצעות הפונקציה CreateMutex. משתמשים בפונקציה CreateMutex כמו שניתן לראות בהגדרה שלהלן:

```
HANDLE CreateMutex(
  LPSECURITY_ATTRIBUTES lpMutexAttributes,
                          // security attributes
  BOOL bInitialOwner,     // flag for initial ownership
  LPCTSTR lpName          // pointer to mutex-object name
);
```

הפרמטר lpMutexAttributes הוא מצביע למבנה מסוג SECURITY_ATTRIBUTES שקובע אם תהליכי בן יכולים לרשת את הידית המוחזרת. אם lpMutexAttributes הוא NULL, תהליך הבן אינו יכול לרשת את הידית. במערכת Windows NT, האיבר lpSecurityDescriptor של איבר המבנה מגדיר מתאר אבטחה עבור המוטציה החדשה. אם lpMutexAttributes הוא NULL, המוטציה מקבלת מתאר אבטחה של ברירת המחדל. במערכת Windows 9x, הפונקציה CreateMutex מתעלמת מאיבר המבנה lpSecurityDescriptor. הפרמטר bInitialOwner מגדיר את הבעלים הראשון של אובייקט המוטציה. אם ערכו אמת, המטלה הקוראת מבקשת בעלות מיידית על אובייקט המוטציה; אחרת, אין למטלה אחרת כלשהי בעלות על אובייקט המוטציה. הפרמטר lpName מצביע למחרוזת המסתיימת ב-NULL שמגדירה את שם אובייקט המוטציה. Windows מגבילה את השם ל-MAX_PATH תווים; השם יכול להכיל כל תו פרט ל**לוכסן הפוך** (\, Backslash) שמשמש מפריד בנתיב תיקיות. זכור תמיד שהשוואת השמות הינה **תלויית רישיות** (Case Sensitive), כלומר יש הבחנה בין אותיות רגילות לרישיות.

✍ **הערה:** המונח **תלוי רישיות** מציין שיש הבחנה בין אותיות לטיניות קטנות לבין אותיות לטיניות רישיות ("גדולות"). תוכניות רבות אינן רגישות להבחנה זו, ואז ניתן להקליד את הפקודה בכל אות רצויה. לדוגמה, בפקודת החיפוש של Windows9x אין הבחנה, אלא אם מציינים במפורש (אפשרויות, תלוי רישיות).

אם הפרמטר lpName חופף לשם אובייקט מוטציה שקיים כבר, lpName מבקש גישה MUTEX_ALL_ACCESS לאובייקט הקיים. הפונקציה CreateMutex מתעלמת מהפרמטר bInitialOwner מכיון שהתהליך היוצר כבר קבע אותו. אם הפרמטר lpMutexAttributes אינו NULL, CreateMutex מחליטה אם אפשר לרשת את הידית, אבל מתעלמת מהידית של איבר מתאר האבטחה. אם lpName הוא NULL, הפונקציה CreateMutex יוצרת את

אובייקט המוטציה בלי שם. אם השם שמוגדר על ידי lpName חופף לשם קיים של **אירוע**, **סמפור**, או אובייקט **קובץ מיפוי** (File Mapping), הפונקציה נכשלת, ואז הפונקציה GetLastError מחזירה את הקבוע ERROR_INVALID_HANDLE, המאפשר גישה אל הידית. הפונקציה נכשלת מכיוון שאובייקטי אירוע, מוטציה, סמפור, וקבצי מיפוי משתפים באותו **מרחב השמות** (Name Space).

לידית המוחזרת על ידי CreateMutex יש גישה MUTEX_ALL_ACCESS אל אובייקט המוטציה החדש, ואפשר להשתמש בה בתוכניות בכל פונקציה שדורשת ידית לאובייקט מוטציה. כל מטלה בתהליך יכולה להגדיר את ידית אובייקט המוטציה בקריאה לאחת מפונקציות ההמתנה (כמו WaitForSingleObject ו-WaitForMultipleObjects). פונקציות ההמתנה של אובייקט יחיד חוזרות כאשר מצב האובייקט המוגדר הוא מסומן. אפשר להורות לפונקציות ההמתנה של אובייקטים רבים לחזור, כאשר אובייקט אחד כלשהו, או כאשר כל האובייקטים המוגדרים נמצאים במצב מסומן. כאשר פונקציית המתנה חוזרת, מערכת ההפעלה משחררת את המטלה הממתינה כדי שתמשיך בפעולה שלה.

המצב של אובייקט מוטציה הוא מסומן, כאשר אין למטלה כלשהי בעלות עליו. המטלה היוצרת יכולה להשתמש בדגל bInitialOwner כדי לבקש מיד בעלות על המוטציה. אחרת, המטלה חייבת להשתמש באחת מפונקציות ההמתנה, כדי לבקש בעלות. כאשר מצב המוטציה מסומן, מערכת ההפעלה מוסרת בעלות לאחת המטלות הממתינות, מצב המוטציה משתנה לאינו מסומן, ופונקציית ההמתנה חוזרת. רק מטלה אחת יכולה לקבל בעלות על המוטציה בכל זמן נתון. המטלה שיש לה בעלות משתמשת בפונקציה ReleaseMutex כדי להשתחרר מבעלות זו. המטלה שהיא הבעלים של המוטציה יכולה להגדיר את אותה מוטציה בקריאות חוזרות לפונקציית ההמתנה, מבלי להינעל. ככלל, מטלה אינה ממתינה בעת קריאות חוזרות לאותה מוטציה, אבל טכניקת הקריאות החוזרות מונעת ממטלה לנעול את עצמה בזמן שהיא ממתינה למוטציה שיש לה כבר בעלות עליה. כדי לשחרר את הבעלות, המטלה חייבת לקרוא לפונקציה ReleaseMutex עבור כל מקרה שבו המוטציה ציינה המתנה.

שני תהליכים או יותר יכולים לקרוא ל-CreateMutex כדי ליצור אובייקט מוטציה בעל אותו שם. התהליך הראשון יוצר את המוטציה, ותהליכים עוקבים פותחים ידית למוטציה הקיימת. תהליכים רבים יכולים להשתמש ב-CreateMutex כדי לאפשר להם לקבל ידיות לאותו אובייקט מוטציה, וכך הם משחררים את המשתמש מהאחריות לוודא שהתהליך היוצר התחיל לפני כן במוטציה. כאשר משתמשים ב-CreateMutex בתהליכים רבים, צריך לקבוע את הדגל bInitialOwner ל-False; אחרת, יהיה קשה לדעת מיהו התהליך הראשון שקיבל את הבעלות. תהליכים רבים יכולים לקבל ידיות של אותו אובייקט מוטציה, דבר שמאפשר לתוכניות להשתמש באובייקט המוטציה עבור **סינכרון פנים תהליכי** (Inter-Process Synchronization). הטכניקות שלהלן ל**שיתוף-אובייקט** (Object-Sharing) זמינות לשימוש התוכניות:

- ☆ תהליך בן שנוצר על ידי הפונקציה CreateProcess יכול לרשת ידית של אובייקט מוטציה, אם הפרמטר lpMutexAttributes של הפונקציה CreateMutex מאפשר ירושה.

✰ תהליך יכול להגדיר את ידית **אובייקט המוטציה** על ידי קריאה לפונקציה DuplicateHandle כדי ליצור ידית משוכפלת, שתהליך אחר יכול להשתמש בה.

✰ תהליך יכול להגדיר את שם אובייקט המוטציה על ידי קריאה לפונקציות CreateMutex או OpenMutex.

השתמש בפונקציה CloseHandle כדי לסגור את הידית. המערכת סוגרת את הידית אוטומטית כאשר התהליך מסתיים. מערכת ההפעלה הורסת את אובייקט המוטציה כאשר המערכת סוגרת את הידית האחרונה אל המוטציה.

16.36 שימוש במוטציה בתוכנית לדוגמה

למדת בסעיף 16.35 שהתוכניות יכולה ליצור בקלות אובייקטי מוטציה, כדי לעזור בסינכרון מטלות דרך תהליכים רבים. בסעיף 16.35 למדת את מהלכי העיבוד הבסיסיים שמאחורי יצירת אובייקט מוטציה. אחרי שיוצרים מוטציה, התוכניות חייבות גם להשיג ידית למוטציה, לפני שהן מסוגלות להשתמש בה בקוד שלהן. למדת שתהליך נוסף יכול להשיג ידית למוטציה על ידי קריאה לפונקציה CreateMutex עם אותו שם מוטציה כמו שעשה התהליך הראשון. באותו אופן, באפשרות התוכניות להשתמש בפונקציה OpenMutex כדי להשיג ידית לאובייקט מוטציה שנוצר קודם. את הפונקציה OpenMutex כותבים בתוכניות, כמו בהגדרה שלהלן:

```
HANDLE OpenMutex(
  DWORD dwDesiredAccess,      // access flag
  BOOL bInheritHandle,        // inherit flag
  LPCTSTR lpName              // pointer to mutex-object name
);
```

הפרמטר dwDesiredAccess מגדיר את הגישה המבוקשת לאובייקט המוטציה. עבור מערכות שתומכות באבטחת אובייקט (כמו בהתקנות המוגנות של Windows NT), הפונקציה נכשלת אם מתאר האבטחה של האובייקט המוגדר אינו מרשה לקבל את הגישה המבוקשת על ידי התהליך הקורא. הפרמטר dwDesiredAccess יכול להכיל אחד משני צירופים: צירוף אחד הוא הערך MUTEX_ALL_ACCESS שמגדיר את כל אפשרויות דגלי הגישה עבור אובייקט המוטציה; צירוף שני הוא הערך SYNCHRONIZE שנתמך רק על ידי Windows NT, ואשר מאפשר לתהליך להשתמש בידית המוטציה בכל אחת מפונקציות ההמתנה כדי לבקש בעלות על אובייקט המוטציה, או בפונקציה ReleaseMutex כדי לשחרר בעלות; או שאפשר להשתמש בשני צירופים אלה.

הפרמטר bInheritHandle מגדיר אם אפשר לרשת את הידית המוחזרת. אם כן, תהליך שנוצר קודם לכן על ידי הפונקציה CreateProcess יכול לרשת את הידית; אחרת, אי אפשר לרשת אותה. הפרמטר lpName מצביע למחרוזת המסתיימת ב-NULL שמכילה את שם אובייקט המוטציה ש-OpenMutex צריכה לפתוח. זכור תמיד, שהשוואת השמות הינה תלויית רישיות.

הפונקציה OpenMutex מאפשרת לתהליכים רבים לפתוח ידיות של אותו אובייקט מוטציה. הפונקציה OpenMutex מצליחה רק אם תהליך מסוים כלשהו יצר כבר את המוטציה על ידי הפונקציה CreateMutex. התהליך הקורא יכול להשתמש בידית המוחזרת בכל הפונקציות שדורשות ידית לאובייקט מוטציה, כמו פונקציות ההמתנה, שמוגבלות להגבלות של הגישה שמוגדרת על ידי הפרמטר dwDesiredAccess.

באפשרות התוכניות להשתמש בפונקציה DuplicateHandle כדי לשכפל ידית ובפונקציה CloseHandle כדי לסגור את הידית. המערכת סוגרת את הידית אוטומטית כאשר התהליך מסתיים. מערכת ההפעלה מפרקת את אובייקט המוטציה כאשר המערכת סוגרת את הידית האחרונה של המוטציה.

כדי להבין יותר טוב את מהלכי העיבוד של התוכניות עם אובייקטי מוטציה, התבונן בתוכנית **Simple_Mutex**, בתקליטור המצורף. התוכנית **Simple_Mutex** יוצרת אובייקט מוטציה פשוט. כאשר המשתמש בוחר באפשרות !Test מהתפריט, התוכנית מתחילה מטלה, אשר ממתינה לגישה למוטציה, נרדמת, ואחר כך משחררת את המוטציה. כאשר המשתמש בוחר מספר פעמים באפשרות !Test מתוך התפריט, התוכנית מציגה **שרשור מוטציות** (Mutex Contention). כלומר, המוטציה מכריחה מטלות עוקבות להמתין עד אשר כל מטלה קודמת משלימה את פעולתה. בזמן שהתוכנית **Simple_Mutex** יוצרת את מטלת הבן בפונקציה WndProc, רוב העיבוד המעשי של התוכנית מתרחש בפונקציה ChildThreadProc.

16.37 השימוש בסמפורים

מרבית הסמפורים משתמשים במונה לשם סינכרון. השימוש בסמפורים מיועד בדרך כלל להגביל גישה לאובייקט, לקטע קוד, או אל משאב מוגבל אחר. כאשר יוצרים סמפור, אומרים לו כמה גישות עליו להרשות, ומהו מספר הגישות ההתחלתי. כדי ליצור סמפור מפעילים את הפונקציה CreateSemaphore, וכותבים אותה כפי שמוצג להלן:

```
HANDLE CreateSemaphore(
  LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
                                // security attributes
  LONG lInitialCount,           // initial count
  LONG lMaximumCount,           // maximum count
  LPCTSTR lpName                // pointer to semaphore name
);
```

הפונקציה CreateSemaphore מקבלת את הפרמטרים שמפורטים בטבלה 16.14.

טבלה 16.14: הפרמטרים של הפונקציה CreateSemaphore.

פרמטר	תיאור
lpSemaphoreAttributes	מצביע למבנה מסוג SECURITY_ATTRIBUTES שקובע אם תהליכי בן יכולים לרשת את הידית המוחזרת. אם lpSemaphoreAttributes שווה ל-NULL, תהליך הבן אינו יכול

פרמטר	תיאור
lpSemaphoreAttributes	לרשת את הידית. תחת Windows NT, האיבר lpSecurityDescriptor של המבנה מגדיר **מתאר אבטחה** (Secrity Descriptor) לסמפור החדש. אם lpSemaphoreAttributes הוא NULL, הסמפור מקבל את מתאר האבטחה של ברירת המחדל. תחת Windows 9x, הפונקציה מתעלמת מהאיבר lpSecurityDescriptor של המבנה.
lInitialCount	מגדיר גודל התחלתי למונה אובייקט הסמפור. ערך זה חייב להיות גדול מ-, או שווה לאפס, וקטן מ-, או שווה ל- lMaximumCount. מצב הסמפור הוא "מסומן" כאשר המונה שלו גדול מאפס; ו"אינו מסומן" כאשר הוא שווה לאפס. פונקציית ההמתנה מקטינה את המונה באחד, כל פעם שמשחררת מטלה אשר המתינה לסמפור. פונקציית ההמתנה מגדילה את המונה בערך מוגדר על ידי הקריאה לפונקציה ReleaseSemaphore.
lMaximumCount	מגדיר את הגודל המקסימלי של המונה עבור אובייקט הסמפור. פרמטר זה חייב להיות גדול מאפס.
lpName	מצביע למחרוזת המסתיימת ב-NULL שמגדירה את שם אובייקט הסמפור. Windows מגבילה את השם ל-MAX_PATH תווים והשם יכול להכיל כל תו חוץ **מלוכסן הפוך** (Backslash) שמשמש מפריד בהצגת נתיב תיקיות (\). זכור תמיד, שהשוואת שמות הינה תלויית רישיות (case sensetive). אם lpName חופף לשם אובייקט סמפור שקיים כבר, lpName מבקש גישה SEMAPHORE_ALL_ACCESS לאובייקט הקיים. הפונקציה מתעלמת מהפרמטרים lInitialCount ו-lMaximumCount מכיון שהתהליך היוצר כבר קבע אותם. אם הפרמטר lpSemaphoreAttributes אינו NULL, הפונקציה מחליטה אם אפשר לרשת את הידית, אך היא מתעלמת מפרמטר מתאר האבטחה. אם lpName שווה ל-NULL, הפונקציה יוצרת את אובייקט הסמפור בלי שם. אם השם שמוגדר על ידי lpName חופף לשם של אירוע קיים, מוטציה או אובייקט קובץ מיפוי (File Mapping), הפונקציה נכשלת והפונקציה GetLastError מחזירה את הקבוע ERROR_INVALID_HANDLE. הפונקציה נכשלת מכיון שאובייקטי אירוע, מוטציה, סמפור וקובץ מיפוי, משתתפים באותו **מרחב שמות** (Name Space).

אם הפונקציה CreateSemaphore מצליחה, הערך המוחזר הוא ידית לאובייקט הסמפור; אם שם אובייקט הסמפור קיים לפני הקריאה לפונקציה, הפונקציה GetLastError מחזירה את הקבוע ERROR_ALREADY_EXISTS; ואם הפונקציה נכשלת, היא מחזירה NULL.

לידית המוחזרת על ידי הפונקציה CreateSemaphore יש גישה מסוג SEMAPHORE_ALL_ACCESS אל אובייקט הסמפור החדש, ואפשר להשתמש בה בתוכניות בכל פונקציה שדורשת ידית לאובייקט סמפור. כל מטלה של תהליך המופעלת יכולה להגדיר את ידית אובייקט הסמפור בקריאה לאחת מפונקציות ההמתנה. פונקציות ההמתנה של אובייקט יחיד חוזרות, כאשר מצב האובייקט המוגדר הוא "מסומן". אפשר להורות בתוכניות לפונקציות ההמתנה של אובייקטים רבים לחזור כאשר אובייקט אחד כלשהו, או כאשר כל האובייקטים המוגדרים, נמצאים במצב מסומן. כאשר פונקציית המתנה חוזרת, מערכת ההפעלה משחררת את המטלה הממתינה כדי שתמשיך בפעולתה.

המצב של אובייקט סמפור הוא "מסומן" (Signaled) כאשר המונה שלו גדול מאפס, והמצב "אינו מסומן" (non-signaled) כאשר המונה שלו שווה לאפס. הפרמטר lInitialCount מגדיר את המונה ההתחלתי. כל פעם אשר מערכת ההפעלה משחררת מטלה ממתינה בגלל מצב הסימון של הסמפור, הסמפור מקטין את המונה באחד. השתמש בפונקציה ReleaseSemaphore כדי להגדיל את מונה הסמפור בערך מוגדר. המונה לעולם אינו יכול להיות קטן מאפס, או גדול יותר מהערך שהוגדר בפרמטר lMaximumCount.

תהליכים רבים יכולים לקבל ידיות של אותו אובייקט סמפור, והדבר מאפשר לתהליכים להשתמש באובייקט עבור סינכרון פנים תהליכי (Inter-Process Synchronization). הטכניקות שלהלן לשיתוף-אובייקט (Object-Sharing) זמינות לשימוש התוכניות:

- ✰ תהליך בן שנוצר על ידי הפונקציה CreateProcess יכול לרשת ידית של אובייקט הסמפור, כאשר הפרמטר lpSemaphoreAttributes של הפונקציה CreateSemaphore מאפשר ירושה.
- ✰ תהליך יכול להגדיר את ידית **אובייקט הסמפור** על ידי קריאה לפונקציה DuplicateHandle, כדי ליצור ידית משוכפלת שתהליך אחר יכול להשתמש בה.
- ✰ תהליך יכול להגדיר שם של אובייקט סמפור על ידי קריאה לפונקציות OpenSemaphore או CreateSemaphore.

השתמש בפונקציה CloseHandle כדי לסגור את הידית. המערכת סוגרת את הידית אוטומטית כאשר התהליך מסתיים. מערכת ההפעלה מפרקת את אובייקט הסמפור כאשר המערכת סוגרת את הידית האחרונה של הסמפור.

למדת בסעיף קודם שהתוכניות משתמשות במוטציה על ידי יצירת המוטציה, פתיחת המוטציה, ושחרור שלה. צעדים דומים נעשים גם עם סמפורים. כדי להבין יותר טוב את פעולת התוכניות עם סמפורים, התבונן בתוכנית **Create_Semaphore**, שנמצאת

בתקליטור המצורף לספר זה (בתיקיה Books\59285). תוכנית זו משתמשת בסמפורים כדי להבטיח שרק ארבע מטלות תוכלנה להפעיל בו-זמנית את פרוצדורת מטלת הבן. בכל פעם שהמשתמש בוחר באפשרות !Test מהתפריט, התוכנית מנסה ליצור מטלה חדשה. פרוצדורת המטלה מוודאת שאפשר להוסיף מטלות (עד ארבע) לפני שהיא מאפשרת למטלה הנוספת להתחיל בעיבוד שלה. אם אין משאבים זמינים לסמפור, פרוצדורת המטלה ממתינה עד שהעיבוד אפשרי. העיבוד המעשי של התוכנית **Create_Semaphore** מתרחש בפונקציה ChildThreadProc.

16.38 עיבוד של אירוע פשוט

כמו שלמדת, אירועים הם הפורמט הפשוט והבסיסי (Primitive) ביותר לסינכרון אובייקטים. בדרך כלל מפעילים אירוע בתוכנית, כדי לסמן למטלה אחת או יותר שפעולה הושלמה. כמו עם סמפורים ומוטציות, גם התוכניות יוצרות אירוע על ידי הקריאה לפונקציה CreateEvent. משתמשים בפונקציה CreateEvent בתוכניות כמו שרואים בהגדרה שלהלן:

```
HANDLE CreateEvent(
  LPSECURITY_ATTRIBUTES lpEventAttributes,
                          // security attributes
  BOOL bManualReset,      // flag for manual-reset event
  BOOL bInitialState,     // flag for initial state
  LPCTSTR lpName          // pointer to event-object name
);
```

הפונקציה CreateEvent מקבלת את הפרמטרים שמפורטים בטבלה 16.15.

טבלה 16.15: הפרמטרים של הפונקציה **CreateEvent**.

פרמטר	תיאור
lpEventAttributes	מצביע למבנה מסוג SECURITY_ATTRIBUTES שקובע אם תהליכי בן יכולים לרשת את הידית המוחזרת. אם lpEventAttributes שווה ל-NULL, תהליך הבן אינו יכול לרשת את הידית. תחת Windows NT, האיבר lpSecurityDescriptor של המבנה מגדיר **מתאר אבטחה** (Secrity Descriprot) לאירוע החדש. אם lpEventAttributes הוא NULL, האירוע מקבל את מתאר האבטחה של ברירת המחדל. תחת Windows 9x, הפונקציה CreateEvent מתעלמת מהאיבר lpSecurityDescriptor של המבנה.
bManualReset	מגדיר אם CreateEvent יוצרת אובייקט אירוע מאופס ידנית או אוטומטית. אם True, חייבים להשתמש בפונקציה ResetEvent כדי לאפס ידנית את המצב לאינו מסומן; אם False, Windows מאפסת אוטומטית את המצב לאינו מסומן אחרי שהמערכת כבר שחררה מטלה ממתינה אחת.

פרמטר	תיאור
bInitialState	מגדיר את המצב ההתחלתי של אובייקט האירוע. אם True, המצב ההתחלתי מסומן; אחרת, אינו מסומן.
lpName	מצביע למחרוזת המסתיימת ב-NULL שמגדירה את שם אובייקט האירוע. Windows מגבילה את השם ל-MAX_PATH תווים, והוא יכול להכיל כל תו חוץ **מלוכסן הפוך** (Backslash) שמשמש מפריד בנתיב (\). זכור תמיד, שהשוואת שמות הינה תלויית רישיות (Case Sensetive). אם lpName חופף לשם אובייקט אירוע שקיים כבר, lpName מבקש גישה EVENT_ALL_ACCESS אל האובייקט הקיים. הפונקציה CreateEvent מתעלמת מהפרמטרים bManualReset ו-bInitialState מכיוון שהתהליך היוצר כבר קבע אותם. אם הפרמטר lpEventAttributes אינו NULL, הפונקציה קובעת אם אפשר לרשת את הידית, אך היא מתעלמת מפרמטר מתאר האבטחה. אם lpName שווה ל-NULL, הפונקציה CreateEvent יוצרת את אובייקט האירוע בלי שם. אם השם שמוגדר על ידי lpName חופף לשם של סמפור קיים, מוטציה או אובייקט קובץ מיפוי (File Mapping), הפונקציה נכשלת והפונקציה GetLastError מחזירה את הקבוע ERROR_INVALID_HANDLE. הפונקציה נכשלת מכיוון שאירוע, מוטציה, סמפור, וקבצי מיפוי משתתפים באותו **מרחב שמות** (Name Space).

אם הפונקציה CreateEvent מצליחה, הערך המוחזר הוא ידית לאובייקט האירוע. אם שם אובייקט האירוע קיים לפני הקריאה לפונקציה, הפונקציה GetLastError מחזירה את הקבוע ERROR_ALREADY_EXISTS; אם הפונקציה נכשלת, הפונקציה מחזירה NULL.

לידית המוחזרת על ידי CreateEvent יש גישה מסוג EVENT_ALL_ACCESS אל אובייקט האירוע החדש, וכל פונקציה שדורשת ידית לאובייקט אירוע יכולה להשתמש בו. כל מטלה של תהליך המופעלת יכולה להגדיר את ידית אובייקט האירוע בקריאה לאחת מפונקציות ההמתנה. פונקציות ההמתנה של אובייקט יחיד חוזרות, כאשר מצב האובייקט המוגדר הוא מסומן. אפשר להורות לפונקציות ההמתנה של אובייקטים רבים לחזור כאשר אובייקט אחד כלשהו, או כאשר כל האובייקטים המוגדרים נמצאים במצב מסומן. כאשר פונקציית המתנה חוזרת, מערכת ההפעלה משחררת את המטלה הממתינה כדי שתמשיך בפעולתה.

הפרמטר bInitialState מגדיר את המצב ההתחלתי של אובייקט האירוע. השתמש בפונקציה SetEvent כדי לקבוע מצב של אובייקט אירוע למסומן. השתמש בפונקציה ResetEvent כדי לאפס את המצב של אובייקט אירוע לאינו מסומן. כאשר המצב של

אירוע המאופס ידנית הוא מסומן, הוא נשאר במצב זה עד אשר הפונקציה ResetEvent מאפסת אותו באופן מכוון למצב אינו מסומן. התוכניות יכולות לשחרר כל מספר של מטלות ממתינות, או מטלות אשר בצורה עקבית התחילו בפעולות המתנה לאובייקט אירוע מוגדר, בזמן שמצב האירוע הוא מסומן.

כאשר המצב של אובייקט אירוע שמאופס אוטומטית הוא מסומן, האובייקט נשאר במצב זה עד אשר התוכנית או מערכת ההפעלה משחררות מטלה ממתינה אחת; ואז, המערכת מאפסת אוטומטית את המצב לאינו מסומן. אם אין מטלות שממתינות, המצב של אובייקט האירוע נשאר מסומן. תהליכים רבים יכולים לקבל ידיות של אותו אובייקט אירוע, דבר שמאפשר לתהליכים להשתמש באובייקט עבור סינכרון פנים תהליכי. הטכניקות שלהלן לשיתוף-אובייקט זמינות לשימוש התוכניות:

- תהליך בן שנוצר על ידי הפונקציה CreateProcess יכול לרשת ידית של אובייקט אירוע, כאשר הפרמטר lpEventAttributes של הפונקציה CreateEvent מאפשר ירושה.
- תהליך יכול להגדיר את ידית אובייקט האירוע על ידי קריאה לפונקציה DuplicateHandle, כדי ליצור ידית משוכפלת, אשר תהליך אחר יכול להשתמש בה.
- תהליך יכול להגדיר את שם אובייקט האירוע על ידי קריאה לפונקציות CreateEvent או OpenEvent.

השתמש בפונקציה CloseHandle כדי לסגור את הידית. המערכת סוגרת את הידית אוטומטית, כאשר התהליך מסתיים. מערכת ההפעלה מפרקת את אובייקט האירוע כאשר המערכת סוגרת את הידית האחרונה של האירוע.

כדי להבין יותר טוב את פעולת התוכניות שמשתמשות באירועים, התבונן בתוכנית **Three_Options**, שבתקליטור המצורף לספר זה (בתיקיה Books\59285). כדי לראות את השפעת הסינכרון באופן הטוב ביותר, הפעל את התוכנית **Three_Options** שלוש פעמים ופרוש את חלונות המופעים על שולחן העבודה. בשני המופעים הראשונים בחר באפשרויות תפריט Read ובמופע השלישי בחר באפשרויות תפריט Write. בשעה שאת פעולות הקריאה אפשר לבצע במקביל, בו-זמנית, כל פעולת כתיבה חייבת להמתין עד שפעולות הקריאה מסתיימות. כלומר, לאחר שפעולות הקריאה מסתיימות, פעולת הכתיבה יכולה להתבצע. אם אתה הופך את הסדר, ובוחר תחילה בפעולת הכתיבה, פעולות הקריאה חייבות להמתין עד שפעולת הכתיבה מסיימת את העיבוד שלה. כל תהליך מציג את המצב הנוכחי שלו בשורת המצב (Status Bar) של החלון. התוכנית **Three_Options** מבצעת את העיבוד המעשי באמצעות הפונקציה WndProc.

פרק 17

קלט/פלט בחלונות

17.1 פעולות קלט/פלט בקבצים ב-Windows (Windows File I/O)

בוודאי מוכרות לך פעולות קלט/פלט שניתן לבצע בקבצים, בעת תכנות בשפת C או בשפת ++C. בסעיפים הבאים תלמד על יסודות פעולות קלט/פלט בקבצים מערכת ההפעלה Windows.

התפיסה המקובלת היא לראות את הקובץ כבלוק (גוש) של נתונים שנמצאים בהתקן אחסון. מזהה מיוחד שידוע כ"שם הקובץ" מאפשר לך לזהות גוש נתונים זה. בסביבת DOS התוכניות שומרות בדרך כלל את הקבצים בדיסקטים ובכונני דיסק קשיח (נכנה אותם בשם הכולל "דיסק"). בסביבת העבודה Windows אנו רואים שלצורך קלט/פלט, Win32 API מתייחס לצינורות בעלי שם (Named Pips), משאבי תקשורת, התקני דיסק, קלט או פלט של קונסול, או לקובץ הדיסק הקלסי כאל "**קובץ**". כל סוגי הקבצים השונים האלה הינם דומים ברמה הבסיסית, אך לכל סוג קובץ יש מאפיינים והגבלות משלו. פונקציות הקבצים של Win32 API מאפשרות לתוכניות לגשת לקבצים ללא תלות במערכת הקבצים שפועלת, או בסוג ההתקן הפיסי. עם זאת, היכולות של קובץ משתנות ממערכת קבצים אחת למערכת קבצים אחרת ומסוג התקן אחד למשנהו.

17.2 צינורות (pipes), משאבים (resources), התקנים (devices) וקבצים (files)

בסעיף קודם למדת ש-Windows תומכת בקלט ופלט של קבצים במגוון התקנים. בסעיפים העוקבים תלמד לא רק על קלט ופלט של קובץ רגיל שמאוחסן בדיסק, אלא תלמד גם כן חלק מיסודות הקלט והפלט בקבצים להתקנים אחרים. טוב להכיר חלק

מההתקנים הנפוצים ביותר שנפגוש בסביבת העבודה של תוכניות Windows. טבלה 17.1 מפרטת כמה מההתקנים הנפוצים ביותר.

טבלה 17.1: התקנים נפוצים והשימוש שלהם.

התקן	השימוש הנפוץ ביותר
File (קובץ)	אחסנת נתונים עקבית.
Directory (ספריה, תיקיה)	מאפיינים ודחיסת קבצים.
Logical Disk Drive (כונן דיסק לוגי)	**פורמט** (Formatting).
Physical Disk Drive (כונן דיסק פיסי)	**טבלת גישה למחיצות** (Partition Table Access).
Serial Port (יציאה טורית)	שידור נתונים דרך קו טלפון.
Parallel Port (יציאה מקבילית)	שידור נתונים למדפסת.
Mailslot (חריץ דואר)	שידור **אחד-לרבים** (One-To-Many) של נתונים, בדרך כלל **ברשת** (Network) למחשב מבוסס Windows (Windows-Based Machine).
Named pipe (צינור בעל שם)	שידור **אחד-לאחד** (One-To-One) של נתונים, בדרך כלל **ברשת** (Network) למחשב מבוסס Windows (Windows-based machine).
Anonymous pipe (צינור אנונימי)	שידור **אחד-לאחד** (One-To-One) של נתונים במחשב יחיד (לעולם לא דרך רשת).
Socket (שקע)	שידור בשיטת **צרור נתונים** (Datagram) או ברצף של נתונים, בדרך כלל ברשת למחשב כלשהו שתומך בשקעים (יכול להיות שהמחשב אינה מריץ את Windows).
Console (קונסול)	חוצץ מסך בחלון טקסט (A Text Window Screen Buffer).

כמו שתגלה, Win32 מנסה לטשטש את השוני בין ההתקנים מפני המשתמש ככל האפשר. במילים אחרות, אם אתה פותח חריץ דואר וקובץ, Windows מאפשרת לך לקרוא מ- ולכתוב ל- לשני ההתקנים עם פונקציות דומות.

עם זאת, צריך לשים לב שמלבד הפונקציות CreateFile, ReadFile ו- WriteFile, Windows מספקת אוסף מקיף מאוד של פונקציות ייחודיות להתקן (Device-Specific Functions), שמאפשרות לתוכנית לנהל את תכונות ההתקן. לדוגמה, אין היגיון לקבוע קצב שידור (Baud Rate) כאשר משתמשים בצינור בעל שם לתקשורת, למרות שנשמע

הגיוני מאוד לעשות כך כאשר משתמשים ביציאת תקשורת. לכן, הפונקציה SetCommConfig תפעל עם התקן יציאה טורית, אך לא תפעל בצורה נכונה עם צינור בעל שם. מסיבה זאת, מרבית הסעיפים הבאים מתמקדים בשימוש כללי של הפונקציות CreateFile, ReadFile ו- WriteFile, ולא ביישום ייחודי לפונקציה של התקן נתון. התבונן בתיעוד של המהדר שלך ושל ההתקן, כדי לקבל מידע נוסף וייחודי אודות התקשורת עם כל התקן שהוא.

17.3 הפונקציה CreateFile לפתיחת קבצים

ממשק התכנות Win32 API תומך בסוגים רבים של התקנים ומאפשר לתוכניות לנהל קבצים באופן נורמלי בכל סוגי ההתקנים. כדי ליצור קובץ בהתקן כלשהו, משתמשים בדרך כלל בפונקציה CreateFile של Win32 API. הפונקציה CreateFile יוצרת או פותחת את האובייקטים הבאים, ומחזירה ידית אשר התוכניות יכולות להשתמש בה כדי לגשת לאובייקט:

- ✰ קבצים.
- ✰ צינורות.
- ✰ חריצי דואר.
- ✰ משאבי תקשורת.
- ✰ התקני דיסק (רק ב- Windows NT).
- ✰ קונסול.
- ✰ ספריות / תיקיות (רק פתיחה).

את הפונקציה CreateFile כותבים בתוכניות כמו בהגדרה שלהלן:

```
HANDLE CreateFile(
  LPCTSTR lpFileName              // pointer to name of the file
  DWORD dwDesiredAccess,          // access (read-write) mode
  DWORD dwShareMode,              // share mode
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                                  // security attributes
  DWORD dwCreationDistribution,   // how to create
  DWORD dwFlagsAndAttributes,     // file attributes
  HANDLE hTemplateFile            // handle to template file
);
```

הפונקציה CreateFile מספקת לתוכניות שליטה משמעותית על הקובץ שיצרת. טבלה 17.2 מפרטת את הפרמטרים של הפונקציה CreateFile.

טבלה 17.2: הפרמטרים של הפונקציה **CreateFile**.

פרמטר	תיאור
lpFileName	מצביע למחרוזת המסתיימת ב-NULL אשר מגדירה את שם האובייקט (קובץ, צינור, חריץ דואר, משאב תקשורת, התקן דיסק, קונסול, או ספריה/תיקיה) אשר הפונקציה CreateFile צריכה ליצור או לפתוח. אם lpFileName הוא נתיב, יש גבול ברירת מחדל לגודל המחרוזת ששווה ל- MAX_PATH תווים. גבול זה קשור לדרך שבה הפונקציה CreateFile מנתחת את הנתיבים (parses paths).
DwDesiredAccess	מגדיר את סוג הגישה לאובייקט. היישום יכול להשיג גישה לקריאה, גישה לכתיבה, גישה לקריאה-כתיבה, או גישה לשאילתה להתקן (Device-Query). פרמטר זה יכול להיות צירוף כלשהו של הערכים שמפורטים בטבלה 17.3 .
dwShareMode	קבוצת דגלי סיביות, אשר מגדירים כיצד התוכניות יכולות לשתף גישה לאובייקט. אם dwShareMode שווה לאפס, התוכניות אינן יכולות לשתף גישה לאובייקט. הניסיונות העוקבים לפתוח את האובייקט נכשלים, עד שהתוכנית המשתמשת באובייקט סוגרת את הידית שלו. כדי לשתף גישה לאובייקט צריך להשתמש בצירוף של דגל אחד או יותר מהערכים שמפורטים בטבלה 17.4.
lpSecurityAttributes	מצביע למבנה מסוג SECURITY_ATTRIBUTES אשר קובע אם תהליכי בן יכולים לרשת את הידית המוחזרת. אם lpSecurityAttributes הוא NULL, תהליכי בן אינם יכולים לרשת את הידית. Windows NT : האיבר lpSecurityDescriptor של המבנה מגדיר מתאר אבטחה (Security-Descriptor) עבור האובייקט. אם lpSecurityAttributes הוא NULL, האובייקט מקבל מתאר אבטחה של ברירת המחדל. מערכת הקבצים של קובץ המטרה חייבת לתמוך באבטחת קבצים וספריות כדי שתהיה השפעה לפרמטר זה על קבצים. Windows 9x : CreateFile מתעלמת מהאיבר lpSecurityDescriptor.

פרמטר	תיאור
dwCreationDistribution	מגדיר את הפעולה שיש לבצע על קבצים קיימים, ואיזו פעולה לנקוט אם הקבצים אינם קיימים. פרמטר זה חייב להיות אחד מהערכים שמפורטים בטבלה 17.5.
dwFlagsAndAttributes	מגדיר את מאפייני הקובץ ודגלים עבור הקובץ. כל צירוף כלשהו של המאפיינים שמפורטים בטבלה 17.6 הם ערכים תקפים עבור הפרמטר dwFlagsAndAttributes, פרט לכך שכל שאר מאפייני הקובץ עוקפים (override) את FILE_ATTRIBUTE_NORMAL. אם הפונקציה CreateFile פותחת את צד הלקוח של צינור בעל שם, אז הפרמטר dwFlagsAndAttributes יכול להכיל גם כן מידע אבטחה לאיכות שירות - SQOS (Security Quality Of Service). סעיף 17.4, מסביר על הפתיחה של צינורות בעלי שם בפירוט. כאשר היישום הקורא מגדיר את הדגל SECURITY_SQOS_PRESENT, הפרמטר dwFlagsAndAttributes יכול להכיל אחד או יותר מהערכים שמפורטים בטבלה 17.7
hTemplateFile	מגדיר ידית עם גישה GENERIC_READ אל תבנית קובץ (Template File). תבנית הקובץ מספקת מאפייני קובץ ומאפיינים מורחבים עבור קובץ שהפונקציה CreateFile יוצרת. תחת Windows 9x, ערך זה חייב להיות NULL. אם מספקים ידית תחת Windows 9x, הקריאה תיכשל.

כמו שלמדת בטבלה 17.2, התוכניות יכולות להגדיר באמצעות הפרמטר dwAccess את רמת הגישה הרצויה לקבצים אשר פותחים עם הפונקציה CreateFile. הערכים האפשריים עבור הפרמטר dwAccess מוצגים בטבלה 17.3.

טבלה 17.3: הערכים האפשריים עבור האיבר **dwAccess**.

ערך	פירוש
0	מגדיר גישת שאילתה-להתקן עבור האובייקט. באפשרות היישום לברר את מאפייני ההתקן מבלי לגשת אליו.
GENERIC_READ	מגדיר גישת קריאה לאובייקט. תוכניות יכולות לקרוא נתונים מהקובץ וקריאות API יכולות להעביר את מצביע הקובץ. כשמחברים ערך זה עם GENERIC_WRITE מקבלים גישת קריאה-כתיבה.

ערך	פירוש
GENERIC_WRITE	מגדיר גישת כתיבה לאובייקט. תוכניות יכולות לכתוב נתונים לקובץ וקריאות API יכולות להעביר את מצביע הקובץ. כשמחברים ערך זה עם GENERIC_READ מקבלים גישת קריאה-כתיבה.

בנוסף **לרמות הגישה** (Access Levels), באפשרות התוכניות שלך להגדיר מצב משותף עבור הקובץ על ידי השימוש בפרמטר dwShareMode. טבלה 17.4 מפרטת את הערכים האפשריים עבור הפרמטר dwShareMode.

טבלה 17.4: הערכים האפשריים עבור הפרמטר **dwShareMode**.

ערך	פירוש
FILE_SHARE_DELETE	רק עבור Windows NT : פעולות פתיחה עוקבות של האובייקט מצליחות, רק אם התהליך הפותח דורש **גישה למחיקה** (Delete Access).
FILE_SHARE_READ	פעולות פתיחה עוקבות אל האובייקט מצליחות, רק אם התהליך הפותח דורש **גישה לקריאה** (Read Access).
FILE_SHARE_WRITE	פעולות פתיחה עוקבות אל האובייקט מצליחות, רק אם התהליך הפותח דורש **גישה לכתיבה** (Write Access).

הפרמטר dwCreate קובע את הפעולה ש-Windows צריכה לבצע כאשר הקובץ קיים, או שאינו קיים. הפרמטר חייב להיות אחד הערכים שמפורטים בטבלה 17.5.

טבלה 17.5: הערכים האפשריים עבור הפרמטר **dwCreate**.

ערך	פירוש
CREATE_NEW	יוצר קובץ חדש. הפונקציה נכשלת אם הקובץ המוגדר כבר קיים.
CREATE_ALWAYS	יוצר קובץ חדש. הפונקציה דורסת (Overwrites) את הקובץ אם כבר קיים.
OPEN_EXISTING	פותח את הקובץ. הפונקציה נכשלת אם הקובץ אינו קיים.
OPEN_ALWAYS	פותח את הקובץ, אם הקובץ קיים. אם הקובץ אינו קיים, הפונקציה יוצרת את הקובץ כאילו הפרמטר dwCreationDistribution היה שווה ל- CREATE_NEW.
TRUNCATE_EXISTING	פותח את הקובץ. ברגע שהקובץ נפתח, Windows מצמצמת את הקובץ, כך שהגודל שלו שווה לאפס (0) בתים. התהליך הקורא חייב לפתוח את הקובץ במצב גישה של GENERIC_WRITE לכל הפחות. הפונקציה נכשלת אם הקובץ אינו קיים.

כאשר יוצרים קובץ כלשהו במערכת ההפעלה Windows, מערכת ההפעלה מצמידה לו מאפיינים. באפשרותך להגדיר דגלים ומאפיינים עבור כל קובץ חדש שאתה יוצר. טבלה 17.6 מפרטת את הדגלים והמאפיינים האפשריים.

טבלה 17.6: ערכי הדגלים והמאפיינים האפשריים עבור הפרמטר **dwFlagsAndAttributes**

מאפיין	פירוש
FILE_ATTRIBUTE_ARCHIVE	מסמן את הקובץ עם תכונת **ארכיב** (Archive). היישום משתמש במאפיין זה כדי לסמן את הקבצים ל**גיבוי** (Backup) או **העברה** (Removal).
FILE_ATTRIBUTE_COMPRESSED	הקובץ או הספריה/תיקיה דחוסים. עבור קובץ, פירוש הדבר שכל הנתונים בו דחוסים. עבור ספריה/תיקיה, פירוש הדבר שהדחיסה היא ברירת המחדל עבור קבצים חדשים שנוצרים ועבור ספריות המשנה.
FILE_ATTRIBUTE_HIDDEN	הקובץ נסתר (אינו נכלל ברשימה הרגילה של הספריה/תיקיה).
FILE_ATTRIBUTE_NORMAL	אין לקובץ מאפיינים אחרים שנקבעו. מאפיין זה תקף רק אם התוכנית משתמשת בו לבד, כאשר התוכנית קוראת ל- CreateFile.
FILE_ATTRIBUTE_OFFLINE	הנתונים של הקובץ אינם תקפים מיד. מציין שנתוני הקובץ הועברו פיסית ל**אחסון מופרד** (Offline Storage).
FILE_ATTRIBUTE_READONLY	הקובץ הוא לקריאה בלבד. היישומים יכולים לקרוא את הקובץ, אך אינם יכולים לכתוב בו או למחוק אותו.
FILE_ATTRIBUTE_SYSTEM	הקובץ הוא חלק ממערכת ההפעלה, או שהוא נמצא בשימושה הבלעדי של מערכת ההפעלה.
FILE_ATTRIBUTE_TEMPORARY	התהליך משתמש כרגע בקובץ לאחסון זמני. היישום צריך למחוק קובץ זמני ברגע שהיישום אינו צריך אותו עוד.
FILE_FLAG_WRITE_THROUGH	מורה למערכת לכתוב דרך מטמון ביניים כלשהו ולפנות ישירות לדיסק. Windows יכולה עדיין לבצע פעולות כתיבה במטמון, אך אינה יכולה לרוקן אותו לפי דרישתה.

מאפיין	פירוש
FILE_FLAG_OVERLAPPED	מורה למערכת לאתחל את האובייקט, כך שפעולות עיבוד שנמשכות זמן רב מחזירות ERROR_IO_PENDING. כאשר מערכת ההפעלה מסיימת את הפעולה, Windows קובעת את האירוע שמוגדר במבנה OVERLAPPED למצב מסומן (Signaled). כאשר מגדירים את הדגל FILE_FLAG_OVERLAPPED, הפונקציות ReadFile ו- WriteFile **חייבות** להגדיר מבנה מסוג OVERLAPPED. כלומר, כאשר מגדירים את הדגל FILE_FLAG_OVERLAPPED, היישום **חייב** לבצע קריאה וכתיבה חופפות. כאשר מגדירים את הדגל FILE_FLAG_OVERLAPPED, המערכת אינה מחזיקה את מצביע הקובץ. התהליך הקורא חייב למסור לפונקציות ReadFile ו- WriteFile את מקום הקובץ, כחלק מהפרמטר lpOverlapped (מצביע למבנה מסוג OVERLAPPED). דגל זה מאפשר לתהליך לבצע עם ידית אחת יותר מפעולה אחת בו-זמנית (לדוגמה, פעולת קריאה וכתיבה בו זמנית).
FILE_FLAG_NO_BUFFERING	מורה למערכת לפתוח את הקובץ ללא מאגרי ביניים או מטמון. כאשר דגל זה מחובר עם FILE_FLAG_OVERLAPPED, הדגל מאפשר סינכרון מקסימלי, מפני שהקלט/פלט אינו נשען על פעולות הסינכרון של מנהל הזיכרון. חלק מפעולות הקלט/פלט נמשכות זמן רב יותר, מכיון שמערכת ההפעלה אינה מחזיקה נתונים במטמון. היישום חייב לעמוד במספר דרישות כאשר עובדים עם קבצים שפותחים אותם עם הפרמטר FILE_FLAG_NO_BUFFERING. הגישה לקובץ חייבת להתחיל בהיסט בתים (Byte Offsets) בקובץ, שהוא כפולות שלמות של גודל הסקטור. הגישה לקובץ חייבת להיות עבור מספר בתים שהם כפולות שלמות של גודל הסקטור. לדוגמה, אם גודל הסקטור הוא 512 בתים, בקשות הקריאה והכתיבה של היישום יכולות להיות

מאפיין	פירוש
	512, 1024, או 2048 בתים, אך לא 335, 981, או 7171 בתים. כתובות החוצץ עבור פעולות קריאה וכתיבה חייבות להיות מיושרות לפי כתובות זיכרון שהן כפולות שלמות של גודל הסקטור. אחת הדרכים ליישר חוצצים בכפולות שלמות של גודל הסקטור, היא להשתמש ב- VirtualAlloc כדי להקצות את החוצצים. הפונקציה מקצה זיכרון אשר מיושר בכתובות שהן כפולות שלמות של גודל דף. מכיון שדף זיכרון וגודל סקטור הם חזקה של 2, זיכרון זה מיושר גם כן בכתובות שהן כפולות שלמות של גודל הסקטור. היישום יכול לקרוא לפונקציה GetDiskFreeSpace כדי לקבוע את גודל הסקטור.
FILE_FLAG_RANDOM_ACCESS	מציין שהתהליך ניגש לקובץ בצורה אקראית. המערכת יכולה להשתמש במאפיין זה כציון לביצוע אופטימיזציה למטמון הקבצים עבור קובץ זה.
FILE_FLAG_SEQUENTIAL_SCAN	מציין שהתהליך עומד לגשת לקובץ באופן סדרתי מתחילתו אל סופו. המערכת יכולה להשתמש במאפיין זה לביצוע אופטימיזציה למטמון הקבצים עבור קובץ זה. אם יישום מעביר את מצביע הקובץ לגישה אקראית, יכול להיות שלא יתקיים מטמון אופטימלי; אך עדיין מובטח פעולות נכונות. הגדרת דגל זה יכולה לשפר ביצוע יישומים שמשתמשים בגישה סדרתית כדי לקרוא קבצים גדולים. גם אפשר להבחין בשיפור הביצועים עבור יישומים שקוראים קבצים גדולים רבים באופן סדרתי, אך לפעמים מדלגים על תחומי בתים קטנים.
FILE_FLAG_DELETE_ON_CLOSE	מציין שמערכת ההפעלה עומדת למחוק את הקובץ מיד לאחר שכל הידיות שלו נסגרות, ולא רק הידית שהגדרת ל- File_FLAG_DELETE_ON_CLOSE. פתיחות עוקבות של הקובץ ייכשלו, אלא אם התהליך הקורא משתמש בדגל FILE_SHARE_DELETE.

מאפיין	פירוש
FILE_FLAG_BACKUP_SEMANTICS	רק עבור Windows NT : מציין שהקובץ נפתח או נוצר לגיבוי, או לפעולת שחזור. מערכת ההפעלה מבטיחה שהתהליך הקורא עובר את בדיקות בטיחות הקובץ, כדי לוודא שיש לו אישור מתאים לעשות זאת. האישורים הרלוונטיים הם SE_RESTORE_NAME ו- SE_BACKUP_NAME. באפשרותך לקבוע גם דגל זה, כדי להשיג ידית לספריה/תיקיה. באפשרות התהליך למסור למספר פונקציות של Win32 ידית ספריה, במקום ידית קובץ.
FILE_FLAG_POSIX_SEMANTICS	מציין שהתהליך עומד לגשת לקובץ לפי כללי POSIX. הדבר כולל מתן רשות להציג קבצים עם שמות זהים אך שונים זה מזה בסוג האות בלבד (רגילה או רישית), במערכות קבצים אשר תומכות באפשרות זו. צריך להיות זהירים כאשר משתמשים באפשרות זאת, מכיון שיישומים שנכתבו עבור MS-DOS או Windows אינם מבחינים בהבדל של סוג אות, ולכן לא יוכלו לגשת לקבצים שנוצרו עם דגל זה.

✍ **הערה:** POSIX (Portable Operating System Interface based in UNIX) הוא אוסף של תקנים בינלאומיים לממשקי מערכות הפעלה בסגנון UNIX.

אם הפונקציה מצליחה, היא מחזירה ידית פתיחה לקובץ המוגדר. אם הקובץ המוגדר קיים לפני הקריאה לפונקציה ו- dwCreationDistribution שווה ל-CREATE_ALWAYS או ל- OPEN_ALWAYS, קריאה ל- GetLastError מחזירה ERROR_ALREADY_EXISTS (אפילו אם הפונקציה מצליחה). אם הקובץ אינו קיים לפני הקריאה, GetLastError מחזירה אפס. אם הפונקציה נכשלת, הפונקציה מחזירה INVALID_HANDLE_VALUE.

כמו שנאמר, הצבת אפס עבור dwDesiredAccess מאפשרת ליישום לקבל מידע אודות מאפייני ההתקן, מבלי לגשת אליו. סוג זה של שאילתות יעיל, לדוגמה, כאשר יישום רוצה לדעת את קיבולת הדיסקט האפשרית בכונן ואת שאר הפורמטים שהוא תומך בהם, מבלי שיהיה דיסקט בכונן.

התקליטור שמצורף לספר זה מכיל את התוכנית **First_File**, שיוצרת את הקובץ file.dat וכותבת בו מחרוזת. כאשר המשתמש בוחר באפשרות !Test מהתפריט, התוכנית קוראת חזרה את המחרוזת מהקובץ, ומציגה את המחרוזת בתיבת הודעה (Message Box).

17.4 הפונקציה CreateFile עם התקנים שונים

התוכניות יכולות להשתמש בפונקציה CreateFile כדי ליצור קבצים במגוון התקנים. יש הבדלים משמעותיים בצורה שבה CreateFile פועלת, כתלות בהתקן שהתוכנית פועלת עליו. בפסקאות הבאות מתוארים כמה מהבדלים אלה.

כאשר משתמשים ב-CreateFile כדי ליצור קובץ חדש, הפונקציה מבצעת את הפעולות הבאות:

- ✰ מצרפת את מאפייני הקובץ והדגלים שמוגדרים על ידי dwFlagsAndAttributes עם FILE_ATTRIBUTE_ARCHIVE.
- ✰ קובעת את אורך הקובץ לאפס.
- ✰ מעתיקה את המאפיינים המורחבים מקובץ התבנית (Template File) אל הקובץ החדש, אם מגדירים את הפרמטר hTemplateFile.

כאשר הפונקציה CreateFile פותחת קובץ קיים, היא מבצעת את הפעולות הבאות:

- ✰ מצרפת את דגלי הקובץ שמוגדרים על ידי dwFlagsAndAttributes אל מאפייני הקובץ הקיימים. הפונקציה CreateFile מתעלמת מהמאפיינים שמוגדרים על ידי dwFlagsAndAttributes.
- ✰ קובעת את אורך הקובץ לפי הערך שמוגדר ב- dwCreationDistribution.
- ✰ מתעלמת מהפרמטר hTemplateFile.
- ✰ מתעלמת מהאיבר lpSecurityDescriptor של המבנה SECURITY_ATTRIBUTES אם הפרמטר lpSecurityAttributes אינו NULL. CreateFile אינה משתמשת באיברי המבנה האחרים. האיבר bInheritHandle הוא הדרך היחידה לציין אם תהליך אחר יכול לרשת את ידית הקובץ.

כשאתה מנסה ליצור קובץ בכונן דיסקטים שאין בו דיסקט, או לפתוח קובץ בכונן CD-ROM שאין בו תקליטור, המערכת מציגה תיבת הודעה, שבה היא מבקשת מהמשתמש להכניס דיסקט או תקליטור, בהתאם. כדי למנוע מהמערכת להציג תיבת הודעה זו, צריך לקרוא לפונקציה SetErrorMode עם SEM_FAILCRITICALERRORS. למידע נוסף אודות SetErrorMode, התבונן בתיעוד המקוון של המהדר שלך.

כאשר CreateFile פותחת את **צד הלקוח** (Client End) של צינור בעל שם, הפונקציה משתמשת במופע כלשהו של הצינור שנמצא **במצב האזנה** (Listening State). התהליך הפותח יכול להכפיל את הידית כמספר הפעמים שנדרש, אבל, אחרי ש- CreateFile פתחה את צד הלקוח, לקוח אחר אינו יכול לפתוח את מופע הצינור בעל השם הזה. הגישה (Access) שאתה מגדיר כאשר CreateFile פותחת צינור, חייבת להיות תואמת לגישה אשר הגדרת בפרמטר dwOpenMode של הפונקציה CreateNamedPipe. כאשר

אתה מציין אבטחה בהקשר לפתיחת צינור בעל שם, התוכנית צריכה לציין את אחד או יותר מדגלי האבטחה שמפורטים בטבלה 17.7.

טבלה 17.7: דגלי האבטחה אשר משתמשים בהם כשיוצרים צינור בעל שם.

ערך	פירוש
SECURITY_ANONYMOUS	הקובץ יכיר את הלקוח **ברמה האנונימית** (Anonymous Level).
SECURITY_IDENTIFICATION	הקובץ יכיר את הלקוח **ברמת הזיהוי** (Identification Level).
SECURITY_IMPERSONATION	הקובץ יכיר את הלקוח **ברמת ההיכרות** (Impersonation Level).
SECURITY_DELEGATION	הקובץ יכיר את הלקוח **ברמת ההסמכה** (Delegation Level).
SECURITY_CONTEXT_TRACKING	מצב העקיבה לאחר האבטחה הוא דינמי. אם אינך מגדיר דגל זה, מצב העקיבה אחרי האבטחה הוא סטטי.
SECURITY_EFFECTIVE_ONLY	רק ההיבטים המופעלים של הקשרי האבטחה של הלקוח תקפים עבור השרת. אם אינך מגדיר דגל זה, כל ההיבטים של הקשרי האבטחה של הלקוח תקפים. דגל זה מאפשר ללקוח להגביל את הקבוצות ואת ההרשאות שהשרת יכול להשתמש בהם בזמן זיהוי הלקוח.

כשהפונקציה CreateFile פותחת את קצה הלקוח של חריץ דואר, הפונקציה מחזירה INVALID_HANDLE_VALUE, אם הלקוח של חריץ הדואר מנסה לפתוח חריץ דואר מקומי לפני ששרת חריץ הדואר יצר אותו על ידי הפונקציה CreateMailSlot.

הפונקציה CreateFile יכולה ליצור ידית למשאב תקשורת, כמו היציאה הטורית COM1. עבור משאבי תקשורת, הפרמטר dwCreationDistribution חייב להיות שווה ל-OPEN_EXISTING, והפרמטר hTemplate חייב להיות NULL. באפשרותך להגדיר גישות קריאה, כתיבה, או קריאה-כתיבה, ובאפשרותך לפתוח את הידית עבור קלט/פלט חופף.

במערכת Windows NT באפשרותך להשתמש בפונקציה CreateFile כדי לפתוח כונן דיסק או מחיצה בכונן הדיסק. הפונקציה מחזירה ידית להתקן הדיסק. באפשרות התוכניות להשתמש מאוחר יותר בידית זו עם הפונקציה DeviceIOControl.

הקריאה חייבת לעמוד בדרישות הבאות, כדי שתצליח:

- ☆ חובה שיהיו לתוכנית הקוראת הרשאות ניהול לפעולה, כדי שתצליח בכונן דיסק קשיח.
- ☆ המחרוזת lpFileName צריכה להיות בפורמט \\.\physicaldrivex כדי לפתוח את הדיסק הקשיח x. מספור הדיסקים הקשיחים מתחיל מאפס. לדוגמה, \\.\physicaldrive2 משיג ידית לדיסק הקשיח השלישי שמותקן במחשב.
- ☆ המחרוזת lpFileName צריכה להיות \\.\x: כדי לפתוח את כונן הדיסקטים x או מחיצה x בדיסק הקשיח. לדוגמה, \\.\A משיג ידית לכונן A במחשב, והמחרוזת \\.\C מיועדת לקבלת ידית לכונן C.
- ☆ במערכת Windows9x טכניקה זאת אינה טובה לפתיחת כונן לוגי. ב-Windows9x הגדרת מחרוזת בפורמט זה גורמת ל-CreateFile להחזיר שגיאה.
- ☆ הפרמטר dwCreationDistribution חייב להיות שווה לערך OPEN_EXISTING.
- ☆ כאשר פותחים כונן דיסקט או מחיצה בדיסק הקשיח, עליך לקבוע את הדגל FILE_SHARE_WRITE בפרמטר dwShareMode.

הפונקציה CreateFile יכולה ליצור ידית לקלט קונסול (CONIN$). אם יש לתהליך ידית פתיחה לקלט קונסול כתוצאה מ**ירושה** (Inheritance) או **שכפול** (Duplication), התהליך יכול גם כן ליצור ידית לחוצץ המסך הפעיל (CONOUT$). אתה חייב להצמיד את התהליך הקורא לקונסול שעבר בירושה או לקונסול שהוקצה על ידי הפונקציה AllocConsole. עבור ידיות קונסול, צריך לקבוע את הפרמטרים של הפונקציה CreateFile כמו שמפורט בטבלה 17.8.

טבלה 17.8: ערכי הפרמטרים של הפונקציה **CreateFile** כאשר יוצרים קונסול.

פרמטר	תיאור
lpFileName	השתמש בערך CONIN$ כדי להגדיר קלט קונסול, ובערך CONOUT$ כדי להגדיר פלט קונסול. CONIN$ משיג ידית לחוצץ הקלט של הקונסול, אפילו אם הפונקציה SetStdHandle מנתבת את ידית הקלט הסטנדרטית. כדי להשיג את ידית הקלט הסטנדרטית, השתמש בפונקציה GetStdHandle. CONOUT$ משיג ידית לחוצץ המסך הפעיל, אפילו אם הפונקציה SetStdHandle מנתבת את ידית הפלט הסטנדרטית. כדי להשיג את ידית הפלט הסטנדרטית, השתמש בפונקציה GetStdHandle.
dwDesiredAccess	מיקרוסופט ממליצה להשתמש רק ב- GENERIC_READ | GENERIC_WRITE, אבל התוכניות שלך יכולות להשתמש באחד מהם כדי להגביל את הגישה.

פרמטר	תיאור
dwShareMode	אם התהליך הקורא יורש את הקונסול, או אם תהליך בן צריך להיות מסוגל לגשת לקונסול, פרמטר זה חייב להיות שווה ל: FILE_SHARE_READ | FILE_SHARE_WRITE.
lpSecurityAttributes	אם אתה רוצה שתהליך הבן יירש את הקונסול, האיבר bInheritHandle של המבנה SECURITY_ATTRIBUTES חייב להיות True.
dwCreationDistribution	אתה צריך להגדיר OPEN_EXISTING כאשר אתה משתמש ב- CreateFile כדי לפתוח את הקונסול.
dwFlagsAndAttributes	מתעלמים ממנו.
hTemplateFile	מתעלמים ממנו.

טבלה 17.9, מציגה את ההשלכות של הצבות שונות של dwDesiredAccess ושל lpFileName, כאשר מציבים בפרמטר lpFileName את הערך CON.

טבלה 17.9: השפעות של קביעת הגישה כאשר פותחים קונסול.

ערך ההצבה	תוצאה
GENERIC_READ	פותח קונסול לקלט.
GENERIC_WRITE	פותח קונסול לפלט.
GENERIC_READ | GENERIC_WRITE	גורם ל- CreateFile להיכשל.

היישום אינו יכול ליצור ספריה/תיקיה עם הפונקציה CreateFile. כדי לעשות זאת, היישום חייב לקרוא ל- CreateDirectory או ל-CreateDirectoryEx. תחת Windows NT באפשרותך לקבוע את הדגל FILE_FLAG_BACKUP_SEMANTICS כדי להשיג ידית לספריה. התוכניות יכולה למסור את ידית הספריה, ולא את ידית הקובץ, למספר פונקציות Win32. חלק ממערכות הקבצים, כמו NTFS, תומך בדחיסה לקבצים נפרדים ולספריות. בפרמטים של כרכים (**כרך** הוא כינוי נוסף לדיסק, או לכונן דיסקים. המונח כרך אינו מרמז דווקא על כונן פיסי, או על כונן לוגי) עבור מערכת קבצים כזאת, ספריה חדשה יורשת את מאפיין הדחיסה של ספריית האב שלה.

17.5 ידיות קבצים

כמו שלמדת בוודאי במערכות ההפעלה DOS ו-UNIX, גם Windows מקצה **ידית קובץ** (File Handle) לכל קובץ שהתוכנית פותחת או יוצרת. ידית קובץ היא מזהה מיוחד, אשר היישום משתמש בו בפונקציות שניגשות לקובץ. ידיות הקובץ נשארות תקפות עד אשר היישום סוגר אותן על ידי הפונקציה CloseHandle, אשר סוגרת את הקובץ

ומרוקנות את החוצץ (Buffer) לדיסק. כאשר היישום מתחיל בפעולתו בפעם הראשונה, הוא יורש את כל ידיות הקבצים הפתוחים מהתהליך אשר הפעיל את היישום, בתנאי שתהליך האב פתח את הקבצים ומרשה ירושה. אם תהליך האב פתח את הקבצים מבלי להרשות ירושה, היישום אינו יורש את ידיות הקבצים הפתוחים.

למדת שבאפשרות היישום לפתוח ידיות קובץ עבור קלט ופלט של הקונסול. אך במקום שם קובץ, היישום מעביר במקרה זה את המחרוזת CONIN$ אל הפונקציה CreateFile כשם קובץ הקלט של הקונסול, ואת CONOUT$ הוא מעביר כשם קובץ הפלט של הקונסול.

17.6 מבט על מצביעי קבצים

כאשר יישום פותח קובץ בפעם הראשונה, מערכת ההפעלה ממקמת בדרך כלל את מצביע הקובץ אל תחילת הקובץ. מצביע הקובץ מסמן את המקום הנוכחי בקובץ, שבו פעולת הכתיבה או הקריאה הבאה תתבצע. ככל שהתוכניות קוראות או כותבות בתים בקובץ, Windows מקדמת את המצביע אל הבית הבא בתור. היישום יכול להעביר באופן מכוון את מקום המצביע אל מקום אחר בקובץ באמצעות הפונקציה SetFilePointer. את הפונקציה SetFilePointer כותבים בתוכניות כמו בהגדרה שלהלן:

```
DWORD SetFilePointer(
  HANDLE hFile,                // handle of file
  LONG lDistanceToMove,        // number of bytes to move
                               // file pointer
  PLONG lpDistanceToMoveHigh,  // address of high-order word
                               // of distance to move
  DWORD dwMoveMethod           // how to move
);
```

הפונקציה SetFilePointer מקבלת את הפרמטרים שמתוארים בטבלה 17.10.

טבלה 17.10: הפרמטרים שמקבלת הפונקציה **SetFilePointer**.

פרמטר	תיאור
hFile	מזהה את הקובץ אשר הפונקציה עומדת להעביר את המצביע שלו. ידית הקובץ חייבת להיות בעלת גישה GENERIC_READ או GENERIC_WRITE לקובץ.
LDistanceToMove	מגדיר את מספר הבתים שהמצביע צריך לעבור. ערך חיובי גורם לדילוג המצביע קדימה, וערך שלילי גורם לדילוג המצביע לאחור.

פרמטר	תיאור
LpDistanceToMoveHeigh	מצביע למילת הסדר הגבוה של המרחק בן ה- 64 סיביות שצריך להעביר. אם הערך של פרמטר זה NULL, SetFilePointer יכולה לעבוד על קבצים שגודלם המקסימלי מגיע ל-(2^{32}-2). אם מגדירים פרמטר זה, גודל הקובץ המקסימלי מגיע ל-(2^{64} -2). פרמטר זה מקבל גם כן את מילת הסדר הגבוה של הערך החדש של מצביע הקובץ.
DwMoveMethod	מגדיר את נקודת ההתחלה לתזוזת מצביע הקובץ. פרמטר זה יכול להיות אחד הערכים שמפורטים בטבלה 17.11.

יש מספר שיטות שונות שאפשר להורות ל-Windows להשתמש בהן, כדי להזיז את מצביע הקובץ למקום רצוי. טבלה 17.11 מפרטת את הערכים המובנים של שיטות העברת מצביע הקובץ.

טבלה 17.11: שיטות העברה אפשריות עבור **SetFilePointer**.

פרמטר	תיאור
FILE_BEGIN	נקודת ההתחלה היא אפס, או התחלת הקובץ. אם הפרמטר FILE_BEGIN מוגדר, הפונקציה מפרשת את lDistanceToMove כמקום שאינו מסומן (Unsigned) עבור מצביע הקובץ החדש.
FILE_CURRENT	הערך הנוכחי של מצביע הקובץ הוא נקודת ההתחלה.
FILE_END	סוף הקובץ הנוכחי הוא נקודת ההתחלה.

אם הפונקציה SetFilePointer מצליחה, היא מחזירה את **ערך הסדר הנמוך של המילה הכפולה** של מצביע הקובץ החדש ; ואם lpDistanceToMoveHeigh אינו NULL, הפונקציה מציבה את **ערך הסדר הגבוה של המילה הכפולה** של מצביע הקובץ החדש במשתנה מסוג long שמוצבע על ידי פרמטר זה. אם הפונקציה נכשלת ו-lpDistanceToMoveHeigh הוא NULL, הפונקציה מחזירה 0xFFFFFFFF.

אם הפונקציה נכשלת ו- lpDistanceToMoveHeigh אינו NULL, הפונקציה מחזירה 0xFFFFFFFF והפונקציה GetLastError מחזירה ערך שונה מ- NO_ERROR.

אינך יכול להשתמש בפונקציה SetFilePointer עם ידית להתקן **שאינו תומך בחיפוש** (Non-Seeking Device), כמו צינור, או התקן תקשורת. כדי לקבוע את סוג הקובץ עבור hFile, השתמש בפונקציה GetFileType. צריך להיות זהירים כאשר קובעים את מצביע הקובץ ביישום מרובה מטלות. יישום אשר המטלות שלו משתתפות בידית קובץ, מעדכנות את מצביע הקובץ, וקריאה מהקובץ חייבת להשתמש **באובייקט קטע קריטי** (Critical Section Object) או **אובייקט מוטציה** (Mutex Object), כדי להגן על עדכון מצביע הקובץ.

כאשר ידית הקובץ hFile נפתחת עם הדגל FILE_FLAG_NO_BUFFERING, היישום יכול להעביר את מצביע הקובץ **למקומות שמיושרים לגבולות סקטורים** (Sector-Aligned Positions) בלבד. מקום שמיושר לגבול סקטור מיוצג על ידי מספר שלם בכפולות של גודל הסקטור. היישום יכול לקרוא לפונקציה GetDiskFreeSpace כדי להשיג את גודל הסקטור. אם יישום קורא ל- SetFilePointer עם ערכי מרחק שצריך לדלג אליהם ושאינם מיושרים לגבולות סקטור, ועם ידית שנפתחה על ידי התוכנית במקור עם FILE_FLAG_NO_BUFFERING - הפונקציה נכשלת, והפונקציה GetLastError מחזירה ERROR_INVALID_PARAMETER.

✍ **הערה:** הפונקציה SetFilePointer דומה לפונקציות lseek ו- fseek.

17.7 הפונקציה WriteFile לכתיבה לקובץ

התוכניות פותחות קבצים באמצעות הפונקציה CreateFile. אם יצרת קובץ עם גישה לכתיבה, התוכנית יכולה אחר כך להשתמש בפונקציה WriteFile כדי לכתוב נתונים לקובץ זה. פונקציה זו מתוכננת לפעולות סינכרוניות ופעולות אסינכרוניות. הפונקציה מתחילה לכתוב נתונים לקובץ מהמקום שמוצבע על ידי מצביע הקובץ. כשהיא מסיימת את הכתיבה, היא מעדכנת את מצביע הקובץ במספר הבתים הממשי שכתבה, מלבד במקרה שהקובץ נפתח עם FILE_FLAG_OVERLAPPED. אם יצרת את ידית הקובץ עבור קלט ופלט (I/O) חופפים, היישום חייב לעדכן את מקום המצביע לאחר סיום הכתיבה.

את הפונקציה WriteFile כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL WriteFile(
  HANDLE hFile,             // handle to file to write to
  LPCVOID lpBuffer,         // pointer to data to write to file
  DWORD nNumberOfBytesToWrite,    // number of bytes to write
  LPDWORD lpNumberOfBytesWritten, // pointer to number of
                                  // bytes written
  LPOVERLAPPED lpOverlapped // pointer to structure
);                                // needed for overlapped I/O
```

הפונקציה WriteFile מקבלת את הפרמטרים שמתוארים בטבלה 17.12.

טבלה 17.12: הפרמטרים שמקבלת הפונקציה **WriteFile**.

פרמטר	תיאור
hFile	מזהה את הקובץ שהפונקציה עומדת לכתוב בו. ידית הקובץ חייבת להיות בעלת גישה GENERIC_WRITE לקובץ.

תיאור	פרמטר
עבור פעולות כתיבה אסינכרוניות ב- Windows NT, פרמטר זה יכול להיות ידית כלשהי שהפונקציה CreateFile פותחת עם הדגל FILE_FLAG_OVERLAPPED, או ידית **שקע** (Socket) שפונקציות **שקע** או **קבלה** (Accept) מחזירות. עבור פעולות כתיבה אסינכרוניות ב- Windows 9x, פרמטר זה יכול להיות ידית של משאב תקשורת, חריץ דואר, או ידית צינור בעל שם, שהפונקציה CreateFile פתחה עם הדגל FILE_FLAG_OVERLAPPED, או ידית **שקע** (Socket) שפונקציות **שקע** או **קבלה** (Accept) מחזירות. Windows 9x אינה תומכת בפעולות כתיבה אסינכרוניות בקבצי דיסק.	
מצביע לחוצץ שמכיל את הנתונים שהפונקציה עומדת לכתוב לקובץ.	lpBuffer
מספר הבתים שצריך לכתוב לקובץ. שלא כמו במערכת ההפעלה MS-DOS, מערכת Windows NT מפרשת ערך של אפס כמגדיר פעולת כתיבה של NULL. פעולת כתיבה NULL איננה כותבת בתים כלשהם, אך גורמת **לחותמת הזמן** (Time Stamp) להשתנות.	nNumberOfBytesToWrite
מצביע למספר הבתים ש-WriteFile כותבת. WriteFile קובעת ערך זה לאפס, לפני שהיא עושה פעולה כלשהי או בדיקת שגיאות. אם ערך lpOverlapped הוא NULL, lpNumberOfBytesWritten אינו יכול להיות NULL ; אם lpOverlapped אינו NULL, אז הפרמטר lpNumberOfBytesWritten יכול להיות NULL. אם זו פעולת כתיבה חופפת, באפשרותך לקרוא ל-GetOverlappedResult כדי לקבל את מספר הבתים שנכתבו. אם hFile קשור עם **יציאת קלט/פלט מסיימת** (I/O Completion Port), באפשרותך לקרוא לפונקציה GetQueuedCompletionStatus כדי לקבל את מספר הבתים שנכתב.	LpNumberOfBytesWritten
מצביע למבנה מסוג OVERLAPPED. הקריאה לפונקציה צריכה מבנה זה, אם התהליך פותח את hFile עם הדגל FILE_FLAG_OVERLAPPED. אם התהליך פתח את hFile עם הדגל FILE_FLAG_OVERLAPPED, הפרמטר lpOverlapped חייב להיות שונה מ- NULL. הוא חייב	lpOverlapped

פרמטר	תיאור
	להצביע למבנה OVERLAPPED תקף. אם hFile נפתח עם FILE_FLAG_OVERLAPPED ו- lpOverlapped הוא NULL, הפונקציה יכול לדווח בצורה לא נכונה שפעולת הכתיבה הושלמה. אם התהליך פתח את hFile עם הדגל FILE_FLAG_OVERLAPPED ו- lpOverlapped אינו NULL, פעולת הכתיבה מתחילה ב**היסט** (Offset) שמוגדר במבנה OVERLAPPED ויכול להיות ש- WriteFile תחזור לפני שמערכת ההפעלה תסיים את הכתיבה. במקרה כזה, WriteFile מחזירה FALSE והפונקציה GetLastError מחזירה ERROR_IO_PENDING. השימוש בקלט/פלט חופפים מאפשר לתהליך הקורא להמשיך בעיבוד בזמן שמערכת ההפעלה מסיימת את הכתיבה. מערכת ההפעלה קובעת את האירוע שהגדרת במבנה OVERLAPPED למצב מסומן בעת שהיא מסיימת את הכתיבה. אם התהליך לא פתח את hFile עם הדגל FILE_FLAG_OVERLAPPED ו- lpOverlapped הוא NULL, הכתיבה מתחילה במקום שבו המצביע מוצב בקובץ הנוכחי ו- WriteFile אינה חוזרת עד שמערכת ההפעלה מסיימת את הפעולה. אם התהליך לא פתח את hFile עם הדגל FILE_FLAG_OVERLAPPED ו- lpOverlapped אינו NULL, הכתיבה מתחילה בהיסט שאתה מגדיר במבנה OVERLAPPED ו- WriteFile אינה חוזרת עד שמערכת ההפעלה מסיימת את הכתיבה.

אם הפונקציה WriteFile מצליחה, הערך המוחזר שונה מאפס; אם היא נכשלת, הפונקציה מחזירה אפס. אם תהליך אחר נועל חלק מהקובץ ופעולת הכתיבה חופפת את החלק שננעל, הפונקציה נכשלת. אסור שיישומים יקראו יכתבו בחוצץ פלט בעת פעולת כתיבה, אלא רק לאחר שהכתיבה מסתיימת. גישה אל הנתונים שבחוצץ הפלט לפני סיום הכתיבה שלהם, יכולה לגרום להשחתת הנתונים שנכתבים מחוצץ זה.

באפשרות התוכניות להשתמש ב-WriteFile עם ידית לפלט קונסול, כדי לכתוב תווים אל חוצץ המסך. מצב העבודה של הקונסול קובע את התנהגות הפונקציה במדויק. הנתונים נכתבים אל מקום הסמן הנוכחי, ומערכת ההפעלה מעדכנת את מקום הסמן בסיום הכתיבה. שלא כמו במערכת ההפעלה MS-DOS, מערכת Windows NT מפרשת הוראה לכתוב אפס בתים, כפעולת כתיבה מסוג NULL, ואז WriteFile אינה מצמצמת

או קוטמת (Truncate) או מרחיבה את הקובץ. כדי לקטום או להרחיב קובץ השתמש בפונקציה SetEndOfFile.

כאשר יישום משתמש בפונקציה WriteFile כדי לכתוב לצינור, פעולת הכתיבה עלולה לא להסתיים אם חוצץ הצינור מלא. במקום זאת, הכתיבה מסתיימת כאשר הקריאה (על ידי הפונקציה ReadFile) מגדילה את החוצץ. אם התהליך סגר כבר את ידית **צינור הקריאה האנונימי** (Anonymous Read Pipe) ו-WriteFile מנסה להשתמש בידית המתאימה לצינור הכתיבה האנונימי כדי לכתוב, הפונקציה מחזירה False ו-GetLastError מחזירה ERROR_BROKEN_PIPE.

הפונקציה WriteFile עלולה להיכשל ולהחזיר ERROR_INVALID_USER_BUFFER או ERROR_NOT_ENOUGH_MEMORY בכל פעם שדרישות קלט/פלט אסינכרוניות רבות ממתינות. כדי לבטל את כל פעולות קלט/פלט האסינכרוניות הממתינות, השתמש בפונקציה CancelIO. פונקציה זו מבטלת רק את הפעולות שהמטלה הקוראת לידית הקובץ המוגדרת מטפלת בהן. פעולות קלט/פלט שמבטלת מערכת ההפעלה כתוצאה מקריאה ל- CancelIO מסתיימות עם הודעת השגיאה ERROR_OPERATION_ABORTED.

אם אתה מנסה לכתוב לכונן דיסקטים שאין בו דיסקט, המערכת מציגה **תיבת הודעה** שמודיעה למשתמש לנסות לחזור על הפעולה. כדי למנוע מהמערכת להציג תיבת הודעה זו, צריך לקרוא לפונקציה SetErrorMode עם SEM_NOOPENFILEERRORBOX. אם hFile הוא ידית לצינור בעל שם, האיברים Offset ו- OffsetHeigh של המבנה OVERLAPPED שמוצבע על ידי lpOverlapped חייבים להיות אפס, או שהפונקציה נכשלת.

התקליטור שמצורף לספר זה מכיל את התוכנית **Write_File** (בתיקיה Books\59285). אשר פותחת את הקובץ file.dat וכותבת מחרוזות לקובץ כאשר התוכנית מתחילה. כאשר המשתמש בוחר !Test, התוכנית כותבת שורת טקסט נוספת לקובץ, ואחר כך מציגה את שתי שורות הטקסט בתיבת הודעה.

17.8 הפונקציה ReadFile לקריאת קובץ

התוכניות משתמשות בפונקציה WriteFile כדי לכתוב לקובץ, ובאופן דומה, הן משתמשות בפונקציה ReadFile כדי לקרוא מקובץ. הפונקציה ReadFile שקוראת נתונים מקובץ, מתחילה לקרוא מהמקום שמוצבע על ידי מצביע הקובץ. אחרי שהפונקציה ReadFile מסיימת את הקריאה, מערכת ההפעלה מעדכנת את מצביע הקובץ במספר הבתים שנקראו בפועל, אלא אם כן התהליך יצר ידית הקובץ עם מאפיין חפיפה. אם התהליך יוצר את ידית הקובץ עבור קלט ופלט (I/O) חופפים, היישום חייב לעדכן את המקום של מצביע הקובץ לאחר סיום הקריאה. את הפונקציה ReadFile כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL ReadFile(
  HANDLE hFile,                 // handle of file to read
```

```
  LPCVOID lpBuffer,               // address of buffer that
                                  // receives data
  DWORD nNumberOfBytesToRead,     // number of bytes to read
  LPDWORD lpNumberOfBytesRead,    // address of number of
                                  // bytes read
  LPOVERLAPPED lpOverlapped       // address of structure
                                  // for data
);
```

הפונקציה ReadFile מקבלת פרמטרים זהים לפרמטרים של הפונקציה WirteFile שמפורטים בסעיף קודם, פרט לכך שהפונקציה ReadFile מצפה לפרמטר nNumberOfBytesToRead במקום הפרמטר nNumberOfBytesToWrite, ומחזירה את הפרמטר lpNumberOfBytesRead במקום הפרמטר lpNumberOfBytesWritten. הפרמטר nNumberOfBytesToRead מגדיר את מספר הבתים ש-ReadFile צריכה לקרוא מהקובץ. הפרמטר lpNumberOfBytesRead מצביע למספר הבתים שנקראו. ReadFile קובעת ערך זה לאפס לפני שהיא עושה פעולה כלשהי, או בודקת שגיאות. אם הפרמטר lpNumberOfBytesRead שווה לאפס כאשר ReadFile מחזירה True בצינור בעל שם, הצד השני של מצב ההודעה בצינור קורא לפונקציה WirteFile עם nNumberOfBytesToWrite שנקבע לאפס.

אם lpOverlapped שווה ל-NULL, lpNumberOfBytesRead אינו יכול להיות NULL. אם lpOverlapped אינו NULL, אז lpNumberOfBytesRead יכול להיות NULL. אם זו פעולת קריאה חופפת, באפשרותך לקרוא ל- GetOverlappedResult כדי לקבל את מספר הבתים שנקראו. אם hFile קשור **ליציאת קלט/פלט משלימה** (I/O completion port), באפשרותך לקרוא ל- GetQueuedCompletionStatus כדי לקבל את מספר הבתים שנקראו.

אם hFile נפתח עם FILE_FLAG_OVERLAPPED, אך lpOverlapped אינו NULL, פעולת הקריאה מתחילה בהיסט שאתה מגדיר במבנה OVERLAPPED. הפונקציה ReadFile עשויה לחזור לפני שפעולת הקריאה הסתיימה. במקרה כזה, ReadFile מחזירה FALSE והפונקציה GetLastError מחזירה ERROR_IO_PENDING. דבר זה מאפשר לתהליך הקורא להמשיך בטיפול בזמן שפעולת הקריאה מסתיימת. מערכת ההפעלה קובעת את האירוע שהגדרת במבנה OVERLAPPED למצב מסומן כאשר הקריאה מסתיימת.

אם hFile לא נפתח עם FILE_FLAG_OVERLAPPED, אך lpOverlapped הוא NULL, פעולת הקריאה מתחילה במקום שמוצבע בקובץ הנוכחי, ו-ReadFile אינה חוזרת עד שפעולת הקריאה מסתיימת. אם התהליך לא פתח את hFile עם FILE_FLAG_OVERLAPPED ו-lpOverlapped אינו NULL, פעולת הקריאה מתחילה בהיסט שאתה מגדיר במבנה OVERLAPPED. הפונקציה ReadFile אינה חוזרת עד שמערכת ההפעלה מסיימת את הקריאה.

כשהפונקציה ReadFile מצליחה, הערך המוחזר שונה מאפס. אם הפונקציה מחזירה ערך שונה מאפס ומספר הבתים שנקראו שווה לאפס, פירוש הדבר שמצביע הקובץ עבר את הסוף הנוכחי של הקובץ בזמן פעולת הקריאה. אך אם התהליך פתח את הקובץ עם FILE_FLAG_OVERLAPPED ו- lpOverlapped אינו NULL, הערך המוחזר של הפונקציה הוא False ו- GetLastError מחזירה ERROR_HANDLE_EOF כאשר מצביע הקובץ מגיע מעבר לסוף הנוכחי של הקובץ. אם הפונקציה נכשלת, הערך המוחזר של הפונקציה הוא אפס.

ReadFile חוזרת כאשר אחד מהדברים שלהלן מתקיים (True): פעולת כתיבה הסתיימה בקצה הכתיבה של צינור, מספר הבתים שצריך לקרוא נקראו כבר, או כשיש שגיאה. אם תהליך אחר נועל חלק מהקובץ ופעולת הקריאה חופפת את החלק שננעל, הפונקציה ReadFile נכשלת. היישומים חייבים להימנע מקריאה או כתיבה לחוצץ הקלט שפעולת קריאה משתמשת בו, עד שפעולת הקריאה מושלמת. גישה לפני הזמן אל חוצץ הקלט עלולה לגרום להשחתת הנתונים שנקראו מחוצץ זה. אפשר לקרוא תווים מחוצץ הקלט של הקונסול על ידי הפונקציה ReadFile עם ידית לקלט הקונסול. מצב העבודה של הקונסול קובע את התנהגות הפונקציה ReadFile במדויק.

אם התהליך קורא מצינור בעל שם במצב הודעה, וההודעה הבאה בתור ארוכה יותר מהערך שמוגדר ב- nNumberOfBytesToRead, הפונקציה ReadFile מחזירה False והפונקציה GetLastError מחזירה ERROR_MORE_DATA. קריאה עוקבת ל- ReadFile או לפונקציה PeekNamedPipe עלולה לגרום לקריאת שארית ההודעה. כאשר אתה קורא מהתקן תקשורת, פסקי הזמן הנוכחיים (שנקבעו והתקבלו על ידי הפונקציות SetCommTimeouts ו-GetCommTimeouts) שולטים בהתנהגות הפונקציה ReadFile. תוצאות שאי אפשר לצפות להן מתרחשות אם אתה נכשל בקביעת ערכי פסק הזמן. אם ReadFile מנסה לקרוא מחריץ דואר, אשר החוצץ שלו קטן מדי, הפונקציה מחזירה False והפונקציה GetLastError מחזירה ERROR_INSUFFICIENT_BUFFER.

אם התהליך סגר כבר את ידית צינור הכתיבה האנונימי (Anonymous Write Pipe) ו- ReadFile מנסה להשתמש בידית המתאימה לצינור הקריאה האנונימי לקרוא, הפונקציה מחזירה False והפונקציה GetLastError מחזירה ERROR_BROKEN_PIPE. הפונקציה ReadFile עלולה להיכשל ולהחזיר ERROR_INVALID_USER_BUFFER או ERROR_NOT_ENOUGH_MEMORY בכל פעם שיש דרישות קלט/פלט אסינכרוניות רבות שממתינות. התוכנית **Write_File** אשר הוצגה בסעיף קודם מראה את השימוש בפונקציה ReadFile.

✍ **הערה:** הקוד הפונקציה ReadFile שבודק את תנאי סוף קובץ (end of, eof file) שונה עבור פעולות קריאה סינכרוניות ואסינכרוניות. כאשר פעולת קריאה סינכרונית מגיעה לסוף קובץ, ReadFile מחזירה True וקובעת את lpNumberOfBytesRead לאפס. כאשר פעולת קריאה אסינכרונית מגיעה לסוף קובץ, הפונקציה ReadFile נכשלת ומחזירה ERROR_HANDLE_EOF.

17.9 סגירת קובץ

השתמשת בידיות קובץ כדי לעבוד עם קבצים מתוך תוכניות Windows. כמו עם שאר הידיות (ידיות זיכרון, או ידיות להקשר התקן), תמיד צריך לסגור את ידית הקובץ לאחר שהתוכנית מסיימת את העיבוד. הפונקציה CloseHandle סוגרת ידית פתוחה של אובייקט. את הפונקציה CloseHandle כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL CloseHandle(HANDLE hObject);
```

הפרמטר hObject מזהה ידית פתוחה של אובייקט. אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; ואם הפונקציה נכשלת, הערך המוחזר הוא אפס.

באפשרותך להשתמש בפונקציה CloseHandle כדי לסגור ידיות של האובייקטים הבאים:

- ☆ קלט או פלט קונסול (Console Input Or Output).
- ☆ קובץ אירוע (Event File).
- ☆ מיפוי קובץ (File Mapping).
- ☆ מוטציה (Mutex).
- ☆ צינור בעל שם (Named Pipe).
- ☆ תהליך (Process).
- ☆ סמפור (Semaphore).
- ☆ מטלה (Thread).
- ☆ Token (רק ב- Windows NT).

הפונקציה CloseHandle מבטלת את תקיפות ידית האובייקט המוגדרת, מפחיתה אחד ממונה ידית האובייקט ומבצעת בדיקות סגירה לאובייקט. לאחר שהתהליך סוגר את הידית האחרונה של אובייקט, מערכת ההפעלה מסירה את האובייקט מטיפול מערכת ההפעלה. הפונקציה CloseHandle אינה סוגרת אובייקטי מודול. סגירת ידית שאינה תקיפה גורם לפתיחת מצב חריג (Exception). הדבר כולל סגירת הידית פעמיים, אי בדיקת הערך המוחזר וסגירת ידית שאינה תקפה, וניסיון להשתמש ב-CloseHandle עם ידית שהוחזרה על ידי FindFirstFile.

✍ **הערה:** השתמש בפונקציה CloseHandle כדי לסגור ידיות שהוחזרו על ידי הפונקציה CreateFile. השתמש ב- FindClose כדי לסגור ידיות שמוחזרות על ידי הפונקציה FindFirstFile.

17.10 שיתוף נתונים עם מיפוי קבצים

מיפוי קבצים (File Mapping) הוא העתקה של תכולת הקובץ אל מרחב הזיכרון הווירטואלי של התהליך. כאשר ממפים קובץ, ההעתק תכולתו ידוע בשם **תצוגת הקובץ** (File View), והמבנה הפנימי שהתוכנית משתמשת בו להחזקת ההעתק ידוע בשם **אובייקט מיפוי קובץ** (File-Mapping Object). כדי לשתף נתונים, תהליך אחר יכול להשתמש באובייקט מיפוי קובץ של התהליך הראשון, כדי ליצור תצוגת קובץ זהה במרחב הזיכרון הווירטואלי שלו.

דוגמה שכיחה לשיתוף נתונים בין תהליכים היא **חילוף נתונים דינמי** (DDE, Dynamic-Data Exchange) ו"האח הצעיר שלו", **קישור והטבעת אובייקטים** (OLE, Object-Linking And Embedding). ב- Windows 95 וב- Windows NT, היישומים צריכים להשתמש במיפוי קבצים. היישומים אינם זקוקים לקובץ ממשי בכדי למפות אותו בזיכרון. היישום יכול להגדיר ערך 0xFFFFFFFF עבור ידית הקובץ בעת קריאה לפונקציה CreateFileMapping, ואז הפונקציה תמפה לזיכרון תצוגה של **קובץ הדפדוף** (Paging File) של מערכת ההפעלה. בסעיף הבא תמצא הסבר מפורט אודות CreateFileMapping.

כאשר ממפים קובץ בזיכרון, התוכניות יכולות למעשה לגשת לנתונים שבו, כאילו הקובץ היה מערך; ומכיון שכך, הן גם יכולות להשתמש בערכי אינדקסים ומצביעים כדי לגשת לאיברי הקובץ, כמקובל במערך.

17.11 מיפוי קובץ בזיכרון הווירטואלי

התוכניות ממפות קבצים במרחב הזיכרון הווירטואלי של התהליך, מכיון שהדבר מאפשר להן לגשת לנתוני הקבצים בצורה יותר יעילה ומהירה. כאשר רוצים למפות קובץ לזיכרון, התוכניות צריכות להשתמש בפונקציה CreateFileMapping כדי ליצור **אובייקט מיפוי קובץ** (File-Mapping Object) עם שם או בלי שם, עבור הקובץ המוגדר. את הפונקציה CreateFileMapping כותבים בתוכניות כמו בהגדרה שלהלן:

```
HANDLE CreateFileMapping(
  HANDLE hFile,             // handle of file to map
  LPSECURITY_ATTRIBUTES lpFileMappingAttributes,
                            // optional security attributes
  DWORD flProtect,          // protection for mapping object
  DWORD dwMaximumSizeHigh,  // object's size, high-order
                            // 32 bits
  DWORD dwMaximumSizeLow,   // object's size, low-order
                            // 32 bits
  LPCTSTR lpName            // name of file-mapping object
);
```

הפונקציה CreateFileMapping מקבלת את הפרמטרים שמפורטים בטבלה 17.13.

טבלה 17.13: הפרמטרים שמקבלת הפונקציה CreateFileMapping.

פרמטר	תיאור
hFile	מזהה הקובץ שצריך ליצור ממנו את אובייקט המיפוי. חייבים לפתוח את הקובץ במצב גישה שמתאים לדגלי ההגנה שמוגדרים על ידי הפרמטר flProtect. מיקרוסופט ממליצה, למרות ש-Windows אינה מחייבת זאת, שתפתח לגישה בלעדית את הקבצים שבכוונתך למפות. אם hFile שווה ל- (HANDLE)0xFFFFFFFF, התהליך הקורא חייב להגדיר גודל אובייקט מיפוי גם בפרמטרים dwMaximumSizeHigh ו- dwMaximumSizeLow. הפונקציה יוצרת אובייקט מיפוי קובץ בגודל שהוגדר על ידי התהליך הקורא, ואשר נתמך על ידי קובץ הדפדוף של מערכת ההפעלה, במקום קובץ בעל שם שנתמך על ידי מערכת הקבצים. תהליכים יכולים להשתתף באובייקט מיפוי קובץ באמצעות שכפול (Duplication), ירושה, או לפי שם.
lpFileMappingAttributes	מצביע למבנה מסוג SECURITY_ATTRIBUTES אשר קובע אם תהליך בן יכול לרשת את הידית המוחזרת. אם lpFileMappingAttributes הוא NULL, תהליכי בן אינם יכולים לרשת את הידית.
flProtect	מגדיר את ההגנה שאתה מבקש עבור תצוגת הקובץ, כאשר הקובץ ממופה ; האפשרות PAGE_READONLY מיועדת לגישת קריאה בלבד אל אזור הדפים המשוריינים (Committed Pages). כוונה כלשהי לכתוב או להפעיל את האזור המשוריין (Committed Region) גורמת לשגיאת גישה (Access Violation). הקובץ אשר מוגדר על ידי הפרמטר hFile חייב להיווצר עם גישה מסוג GENERIC_READ. האפשרות PAGE_READWRITE מיועדת לגישת קריאה וכתיבה לאזור הדפים המשוריינים. הקובץ שמוגדר על ידי הפרמטר hFile חייב להיווצר עם גישות מסוג GENERIC_READ ו- GENERIC_WRITE. האפשרות PAGE_WRITECOPY נותנת העתק בעת גישת כתיבה לאזור הדפים המשוריינים. הקבצים שמוגדרים על ידי הפרמטר hFile חייבים להיווצר עם גישות מסוג GENERIC_READ ו- GENERIC_WRITE. בנוסף לכך,

פרמטר	תיאור
	באפשרות היישום לצרף (על ידי אופרטור הסיביות OR) ערך אחד או יותר מערכי מאפייני הקטע (Section Attribute) שמפורטים בטבלה 17.14 אל אחד משלושת סוגי הגנת הדף שמנינו עד עכשיו, כדי להגדיר מאפייני קטע מסוימים.
dwMaximumSizeHigh	מגדיר את ערך הסדר העליון בן 32 סיביות של הגודל המקסימלי של אובייקט מיפוי הקובץ.
dwMaximumSizeLow	מגדיר את ערך הסדר הנמוך בן 32 סיביות של הגודל המקסימלי לאובייקט מיפוי הקובץ. אם פרמטר זה והפרמטר dwMaximumSizeHigh שווים לאפס, הגודל המקסימלי של אובייקט מיפוי קובץ שווה לגודל הקובץ הנוכחי שמוגדר על ידי הפרמטר hFile.
lpName	מצביע למחרוזת המסתיימת ב-NULL אשר מגדיר את שם אובייקט המיפוי. השם יכול להכיל כל תו מלבד התו לוכסן הפוך (\). אם פרמטר זה חופף לשם קיים של אובייקט מיפוי, הפונקציה דורשת גישה לאובייקט המיפוי עם ההגנה שמוגדרת על ידי flProtect. אם פרמטר זה הוא NULL, הפונקציה יוצרת את אובייקט המיפוי בלי שם.

מתוך טבלה 17.13 למדת שאפשר לצרף את ערכי הגנת הדף עבור הקובץ עם ערך אחד או יותר ממאפייני הקטע, כמו שמוצג בטבלה 17.14.

טבלה 17.14: הערכים של הגנת הדף עבור מיפוי קובץ בזיכרון.

ערך	תיאור
SEC_COMMIT	מקצה אחסון פיסי בזיכרון או בקובץ הדפדוף שבדיסק עבור כל הדפים שבקטע. זוהי ברירת המחדל.
SEC_IMAGE	הקובץ שאתה מגדיר עבור קטע מיפוי קובץ הוא **קובץ תמונה בר-הפעלה** (Executable Image File). מפני שמידע המיפוי והגנת הקובץ נלקחים מקובץ התמונה, מאפיינים אחרים אינם תקפים עם SEC_IMAGE.
SEC_NOCACHE	הפונקציה קובעת שכל הדפים של קטע הם ללא מטמון. ב- 80x86 ומחשבי MIPS, השימוש במטמון למבנים אלה מאיט את הביצוע, ככל שהחומרה שומרת על עדכון המטמון. חלק ממנהלי ההתקנים דורש נתונים ללא מטמון, כך שתוכניות יכולות לכתוב דרך הזיכרון הפיסי. SEC_NOCACHE דורש לקבוע את SEC_RESERVE או SEC_COMMIT כדי שיהיה אפשר לקבוע אותו.

ערך	תיאור
SEC_RESERVE	שומר את כל הדפים של קטע, מבלי להקצות אמצעי אחסון פיסי. פעולות הקצאה אחרות כלשהן אינן יכולות להשתמש בתחום הדפים המוקצה, עד שהוא משוחרר. באפשרות היישום לשריין דפים מוקצים על ידי קריאות עוקבות לפונקציה VirtualAlloc. תכונה זאת תקפה רק אם הפרמטר hFile שווה ל- HANDLE)0xFFFFFFFF); כלומר, אובייקט מיפוי קבצים שנתמך על ידי קובץ הדפדוף של מערכת ההפעלה.

כאשר הפונקציה מצליחה, הערך המוחזר הוא ידית לאובייקט מיפוי קובץ. אם האובייקט הממופה קיים לפני הקריאה לפונקציה, הפונקציה GetLastError מחזירה ERROR_ALREADY_EXISTS, והפונקציה מחזירה ידית תקפה לאובייקט מיפוי הקובץ הקיים (עם הגודל הנוכחי שלו, לא הגודל החדש המוגדר). אם אובייקט המיפוי אינו קיים, GetLastError מחזירה אפס. אם הפונקציה נכשלת, היא מחזירה NULL.

אחרי שתהליך יצר כבר אובייקט מיפוי קובץ, גודל הקובץ חייב להיות קטן מגודל אובייקט מיפוי הקובץ; אם כך המצב, לא כל תכולת הקובץ תהיה תקפה לשימוש משותף של היישומים. אם היישום מגדיר גודל עבור אובייקט מיפוי קובץ שמעבר לגודלו הנוכחי של הקובץ בדיסק, Windows מגדילה את הקובץ בדיסק כדי שיתאים לגודל שהוגדר לאובייקט מיפוי הקובץ. לידית שמוחזרת על ידי CreateFileMapping יש גישה מלאה לאובייקט מיפוי הקובץ. באפשרות היישום להשתמש בידית עם כל פונקציה שדורשת ידית לאובייקט מיפוי קובץ. שיתוף תהליכים יכול להיות באובייקט מיפוי קבצים, דרך יצירת התהליך, על ידי שכפול הידית, או לפי שם.

✍ **הערה:** תחת Windows 9x, אסור להשתמש בידיות קבצים אשר השתמשת בהן, כדי ליצור אובייקטי מיפוי קבצים בקריאות הבאות לפונקציות קלט/פלט, כמו ReadFile ו-WriteFile. ככלל, אם השתמשת בידית קובץ בקריאה שהצליחה לפונקציה CreateFileMapping, אל תשתמש שוב בידית זו, אלא אם קודם אתה סוגר תחילה את אובייקט מיפוי הקובץ המתאים.

יצירת אובייקט מיפוי קובץ יוצרת את הפוטנציאל למפות תצוגה של הקובץ, אך אינה ממפה את התצוגה. הפונקציות MapViewOfFile ו-MapViewOfFileEx ממפות תצוגת קובץ במרחב הזיכרון של התהליך.

ויש יוצא מן הכלל: תצוגות קובץ שנגזרות מאובייקט מיפוי קובץ יחיד הן עקביות (Coherent) בזמן נתון. אם יש לתהליכים רבים ידיות של אותו אובייקט מיפוי קובץ, הם רואים תצוגה עקבית של הנתונים כאשר הם ממפים תצוגת קובץ. היוצא מן הכלל מתייחס לקבצים מרוחקים (Remote Files). למרות ש-CreateFileMapping פועלת עם קבצים מרוחקים, היא אינה שומרת על עקביות שלהם. לדוגמה, אם שני מחשבים

ממפים קובץ כמי ניתן לכתוב בו ושניהם משנים את אותו דף, כל מחשב יראה רק את הכתיבה שהוא עשה בדף. כאשר הנתונים מעודכנים בדיסק, מערכת ההפעלה אינה ממזגת את הנתונים. כמו כן, קובץ ממופה וקובץ אשר פונקציות הקלט/פלט (ReadFile ו- WriteFile) ניגשות אליהם, אינם בהכרח עקביים.

כדי לסגור באופן מלא אובייקט מיפוי קבצים, היישום חייב לקרוא ל- UnmapViewOfFile כדי להפסיק את כל מיפויי התצוגה של אובייקט מיפוי הקובץ, ולקרוא ל- CloseHandle כדי לסגור את ידית אובייקט מיפוי הקובץ. הסדר שבו יישום קורא לפונקציות אלו אינו משנה. יש צורך בקריאה ל- UnmapViewOfFile מפני שתצוגות ממופות של אובייקט מיפוי קובץ מחזיקות ידיות פנימיות פתוחות לאובייקט, ואובייקט מיפוי קובץ אינו נסגר עד שהתהליך סוגר את כל הידיות הפתוחות של אובייקט מיפוי הקובץ.

17.12 מיפוי תצוגת קובץ אל התהליך הנוכחי

כשתוכניות יוצרות אובייקט מיפוי קובץ, הן חייבות למפות את הקובץ באובייקט. כדי לעשות זאת, הן משתמשות בפונקציה MapViewOfFile. פונקציה זו ממפה תצוגת קובץ במרחב הזיכרון של התהליך הקורא.

את הפונקציה MapViewOfFile כותבים בתוכניות כמו בהגדרה שלהלן:

```
LPVOID MapViewOfFile(
  HANDLE hFileMappingObject,  // file-mapping object
                              // to map into address space
  DWORD dwDesiredAccess,      // access mode
  DWORD dwFileOffsetHigh,     // high-order 32 bits
                              // of file offset
  DWORD dwFileOffsetLow,     // low-order 32 bits of file offset
  DWORD dwNumberOfBytesToMap, // number of bytes to map
);
```

הפרמטר hFileMappingObject מציין ידית פתוחה של אובייקט מיפוי קובץ. הפונקציות CreateFileMapping ו- OpenFileMapping מחזירות ידית זו. הפרמטר dwDesiredAccess מגדיר את סוג הגישה לתצוגת הקובץ ולמעשה, את הגנת הדפים שהקובץ ממפה. פרמטר זה יכול להיות אחד הערכים שמפורטים בטבלה 17.15.

הפרמטר dwFileOffsetHigh מגדיר את ערך הסדר העליון בן 32 סיביות של היסט הקובץ בנקודה שבה המיפוי מתחיל; והפרמטר dwFileOffsetLow מגדיר את ערך הסדר הנמוך בן 32 סיביות של היסט הקובץ במקום שבו המיפוי מתחיל. הצירוף של ההיסט הגבוה ושל ההיסט הנמוך חייב להגדיר היסט בקובץ שתואם לצורת הקצאת הזיכרון של המערכת, או שהפונקציה תיכשל. כלומר, ההיסט חייב להיות כפולה של יחידת

ההקצאה (8 בתים, או 16 בתים). הפרמטר dwNumberOfBytesToMap מגדיר את מספר הבתים של הקובץ שצריך למפות, ואם הוא שווה לאפס, מערכת ההפעלה ממפה את כל הקובץ.

טבלה 17.15: הערכים האפשריים עבור הפרמטר **dwDesiredAccess**.

ערך	פירוש
FILE_MAP_WRITE	גישה לקריאה-כתיבה. הפרמטר hFileMappingObject חייב להיות עם ההגנה PAGE_READWRITE. מערכת ההפעלה ממפה את תצוגת הקריאה-כתיבה של הקובץ.
FILE_MAP_READ	גישה לקריאה בלבד. הפרמטר hFileMappingObject חייב להיות עם ההגנה PAGE_READWRITE או PAGE_READONLY. מערכת ההפעלה ממפה את תצוגת הקריאה בלבד של הקובץ.
FILE_MAP_ALL_ACCESS	כמו FILE_MAP_WRITE.
FILE_MAP_COPY	העתק בגישה לקריאה. אם יצרת את המיפוי עם PAGE_WRITECOPY ואת התצוגה עם File_MAP_COPY, אתה מקבל תצוגה אל קובץ. אם אתה כותב בקובץ, הדפים מוחלפים אוטומטית (פעולת Swap), והשינויים שתעשה לא יועברו לקובץ המקור. תחת Windows 9x, אתה חייב למסור את הפרמטר PAGE_WRITECOPY לפונקציה CreateFileMapping ; אחרת, Windows מחזירה שגיאה. תחת Windows NT, אין שום מגבלה לדרך שבה אתה חייב ליצור את הפרמטר hFileMappingObject. העתק כתיבה תקף לכל סוג תצוגה כלשהו. אם אתה משתמש ב- DuplicateHandle או OpenFileMapping כדי לשתף במיפוי תהליכים רבים, ותהליך אחד כותב לתצוגה, השינוי משתקף גם בתהליכים אחרים. הקובץ המקורי אינו משתנה.

אם הפונקציה מצליחה, הפונקציה מחזירה את כתובת ההתחלה של התצוגה הממופה ; אם הפונקציה נכשלת, הפונקציה מחזירה NULL. מיפוי קובץ גורם לחלק הממופה של הקובץ להיראות במרחב הזיכרון של התהליך הקורא.

התקליטור שמצורף לספר זה מכיל את התוכנית **Sample_Mapp** (בתיקיה Books\59285). תוכנית זו יוצרת קובץ מיפוי בזיכרון כאשר היישום מתחיל. כאשר המשתמש בוחר באפשרות !Test, התוכנית ממפה תצוגה לקובץ וממקמת את הנתונים בזיכרון. אחר כך היא יוצרת מטלה וממתינה לקוצב הזמן (Timer) שלפיו המטלה תשנה את הנתונים. המטלה משתמשת בנתונים כדי להציג תיבת הודעה וממקמת את המחרוזת "Received" בזיכרון הממופה כאשר המשתמש סוגר את תיבת ההודעה.

כשהתוכנית מקבלת את ההודעה WM_TIMER, היא בודקת כדי לראות אם המטלה מיקמה את המחרוזת בזיכרון הממופה, או לא. אם כן, התוכנית מפסיקה ומסיימת את המטלה ואת קוצב הזמן ומודיעה למשתמש על כך.

17.13 פתיחת אובייקט קובץ מיפוי בעל שם

התוכניות משתמשות בפונקציה CreateFileMapping כדי ליצור מיפוי קובץ בעל שם, או ללא שם; ובפונקציה MapViewOfFile - כדי למפות את הקובץ בזיכרון. בתוכנית **Sample_Mapp** תוכל לראות שמטלה שנייה פותחת את מיפוי הקובץ. במקום ליצור מיפוי קובץ חדש, המטלה השנייה משתמשת בפונקציה OpenFileMapping כדי לפתוח את מיפוי הקובץ לשימוש של עצמו. את הפונקציה OpenFileMapping כותבים בתוכניות כמו בהגדרה שלהלן:

```
HANDLE OpenFileMapping(DWORD dwDesiredAccess,
                       BOOL bInheritHandle,
                       LPCTSTR lpName);
```

הפרמטר dwDesiredAccess מגדיר את הגישה לאובייקט מיפוי הקובץ. תחת Windows NT, מערכת ההפעלה בודקת את הגישה שיש לפרמטר כנגד כל מתאר אבטחה כלשהו באובייקט המטרה של מיפוי הקובץ. פרמטר זה יכול להיות אחד הערכים שמפורטים בטבלה 17.15. הפרמטר bInheritHandle מגדיר אם תהליך חדש יכול לרשת את הידית המוחזרת בזמן יצירת התהליך. ערך True מציין שהתהליך החדש יורש את הידית. הפרמטר lpName מצביע למחרוזת שנותנת את השם לאובייקט מיפוי הקובץ שהתהליך אמור לפתוח. אם יש ידית פתוחה לאובייקט מיפוי קובץ עם אותו שם, ומתאר האבטחה באובייקט המיפוי אינו מתנגש עם הפרמטר dwDesiredAccess, פעולת הפתיחה מצליחה.

אם הפונקציה מצליחה, הערך המוחזר הוא ידית פתוחה לאובייקט מיפוי הקובץ המוגדר; ואם הפונקציה נכשלת, הפונקציה מחזירה NULL. באפשרותך להשתמש בידית שמוחזרת על ידי OpenFileMapping עם כל פונקציה אשר דורשת ידית לאובייקט מיפוי קובץ.

17.14 תכונות קובץ

Windows מקשרת קבוצה של **תכונות קובץ** (File Attributes) עם כל קובץ. Windows מאתחלת רבים מתכונות הקובץ כאשר יוצרים את הקובץ ויותר מאוחר, Windows משנה חלק מתכונות אלו בכל פעם שניגשים לקובץ. כמעט תמיד, אינך רוצה לשנות את תכונות הקבצים, אלא רק לקרוא אותם ולהגיב לפי הערכים המוצגים. כמו שלמדת בוודאי במערכת ההפעלה DOS, מרבית תכונות הקובץ משתמשות בקביעת דגלים, גודל קובץ, וחותמת הזמן של קובץ (File Time Stamps). בסעיפים הבאים בהמשך, ננהל את תכונות הקובץ וניגש לחלק מהתכונות השימושיות ביותר.

17.15 קבלה ושינוי של תכונות הקובץ

כידוע, Windows מצמידה "תכונות" (Attributes) לכל קובץ שיוצרים במערכת ההפעלה. כמו שאפשר לקרוא את תכונות הקובץ עם פונקציות סטנדרטיות של C, כך גם ממשק Windows API מספק פונקציות רבות שאפשר להשתמש בהן כדי לקרוא מאפייני קובץ. הפונקציה GetFileAttributes מחזירה מידע מתת-קבוצה של כל התכונות האפשריות עבור קובץ מוגדר או ספריה/תיקיה. בסעיפים הבאים נדון בפונקציות אחרות שמחזירות תכונות קובץ אחרות, כמו זמן יצירת הקובץ או גודל הקובץ. את הפונקציה GetFileAttributes כותבים בתוכניות כמו בהגדרה שלהלן:

```
DWORD GetFileAttributes(LPCTSTR lpFileName);
```

הפרמטר lpFileName מצביע למחרוזת המסתיימת ב-NULL, אשר מגדירה שם קובץ או שם ספריה/תיקיה. תחת Windows NT קיים ערך ברירת מחדל של אורך מחרוזת שם הנתיב (Path), שגודלו MAX_PATH תווים. מגבלה זו נובעת מהדרך שבה הפונקציה GetFileAttributes מפרשת את הנתיבים. היישום יכול לקרוא לגירסה הרחבה (Wide) של GetFileAttributes ולהכניס את "\\?\" לנתיב, כדי לעבור את המגבלה הזו, ולהציב בנתיב יותר מ-MAX_PATH תווים. הרצף "\\?\" מורה לפונקציה לכבות את תהליך הפירוש הנתיב; רצף זה מאפשר לתוכניות להשתמש במחרוזות נתיב שארוכות יותר מ-MAX_PATH כשפועלים עם הפונקציה GetFileAttributesW. פונקציה זו גם פועלת עם שמות UNC. מערכת ההפעלה מתעלמת מהרצף "\\?\" שנמצא במחרוזת הנתיב. לדוגמה, הפונקציה רואה את "\\?\C:\mydocuments\private" כ-"C:\mydocuments\private", ואת "\\?\Jamsa1\happy\foodstuff" היא רואה כ-"\\Jamsa1\happy\foodstuff". תחת Windows 9x, אסור שהמחרוזת lpFileName תעבור MAX_PATH תווים.Windows 9x אינה תומכת בקידומת "\\?\".

אם הפונקציה מצליחה, הערך המוחזר מכיל את תכונות הקובץ המוגדר, או הספריה/תיקיה; ואם הפונקציה נכשלת, היא מחזירה 0xFFFFFFFF. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל- GetLastError.

התכונות יכולות להיות אחד הערכים או יותר, מאלה שמפורטים בטבלה 17.16.

טבלה 17.16: הערכים המוחזרים מהפונקציה **GetFileAttributes**.

ערך	פירוש
FILE_ATTRIBUTE_ARCHIVE	הקובץ או הספריה/תיקיה, הוא קובץ ארכיב או ספריית/תיקיית ארכיב בהתאמה. היישומים משתמשים בדגל זה כדי לסמן קבצים לגיבוי או העברה.
FILE_ATTRIBUTE_COMPRESSED	הקובץ או הספריה/תיקיה הם דחוסים. עבור קובץ, פירוש הדבר שכל הנתונים שבו דחוסים. עבור ספריה/תיקיה, פירוש הדבר שהדחיסה היא ברירת המחדל עבור הקבצים החדשים שנוצרים ועבור רמות המשנה.

פירוש	ערך
האובייקט הנוכחי הוא ספריה/תיקיה.	FILE_ATTRIBUTE_DIRECTORY
הקובץ או הספריה/תיקיה נסתרים (אינם נכללים ברשימה הרגילה המוצגת).	FILE_ATTRIBUTE_HIDDEN
אין לקובץ או לספריה/תיקיה תכונות אחרות שנקבעו. תכונה זו תקפה רק אם משתמשים בה לבד.	FILE_ATTRIBUTE_NORMAL
הנתונים של הקובץ אינם תקפים מיד. מציין שנתוני הקובץ הועברו פיסית ל**אחסון מופרד** (Offline Storage).	FILE_ATTRIBUTE_OFFLINE
הקובץ או הספריה/תיקיה הם לקריאה בלבד. היישומים יכולים לקרוא את הקובץ, אך אינם יכולים לכתוב בו או למחוק אותו. במקרה של ספריה/תיקיה, היישום אינו יכול למחוק אותה.	FILE_ATTRIBUTE_READONLY
הקובץ או הספריה/תיקיה הם חלק ממערכת ההפעלה, או שהם בשימוש הבלעדי של מערכת ההפעלה.	FILE_ATTRIBUTE_SYSTEM
תהליך משתמש כעת בקובץ לאחסון זמני. היישום צריך למחוק קובץ זמני ברגע שהיישום אינו צריך אותו עוד.	FILE_ATTRIBUTE_TEMPORARY

תוכניות יכולות להשתמש בפונקציה GetFileAttributes כדי להשיג את תכונות הקובץ, אך לפעמים התוכניות גם צריכות לשנות את התכונות של הקובץ. באפשרותך להשתמש בפונקציה SetFileAttributes כדי לקבוע את תכונות הקובץ, כמו שרואים בהגדרה שלהלן:

```
BOOL SetFileAttributes(
  LPCTSTR lpFileName,          // address of filename
  DWORD dwFileAttributes       // attributes to set
);
```

הפרמטר lpFileName מצביע למחרוזת אשר מגדירה את שם הקובץ שהפונקציה אמורה לקבוע את תכונותיו. המגבלות שחלות על הפרמטר lpFileName של הפונקציה GetFileAttributes חלות גם על הפרמטר lpFileName של הפונקציה SetFileAttributes. הפרמטר dwFileAttributes מגדיר את תכונות הקובץ שהפונקציה צריכה לקבוע. פרמטר זה יכול להיות צירוף של הערכים שמפורטים בטבלה 17.16. עם זאת, כל שאר הערכים עוקפים את FILE_ATTRIBUTE_NORMAL.

אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, הפונקציה מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא לפונקציה GetLastError. אינך יכול להשתמש בפונקציה SetFileAttributes כדי לקבוע את מצב הדחיסה של הקובץ. קביעת FILE_ATTRIBUTE_COMPRESSED בפרמטר dwFileAttributes אינה עושה כלום. השתמש בפונקציה DeviceIoControl ובפעולה FSCTL_SET_COMPRESSION כדי לקבוע את מצב הדחיסה של הקובץ.

בתקליטור המצורף לספר תמצא את התוכנית **Check_ReadOnly** (Books\59285). התוכנית בודקת את הקובץ file.dat כדי לקבוע אם נקבעה לו התכונה "קריאה בלבד". אם כן, התוכנית מוחקת את כל קביעות התכונות, ואחר כך מוחקת את הקובץ.

17.16 איך לקבל את גודל הקובץ

Windows מצמידה תכונות מסוימות לכל קובץ שהיא שומרת בדיסק. אחת הבקשות השכיחות ביותר בתוכניות העוסקות בקבצים היא לדעת את גודל הקובץ. הפונקציה GetFileSize משיגה את הגודל בבתים, של הקובץ המוגדר. את הפונקציה GetFileSize כותבים בתוכניות כבהגדרה שלהלן:

```
DWORD GetFileSize(
  HANDLE hFile,                // handle of file to get size of
  LPDWORD lpFileSizeHigh       // address of high-order
);                             // word for file size
```

הפרמטר hFile מגדיר ידית פתוחה של הקובץ שאמורה הפונקציה להחזיר את הגודל שלו. התהליך חייב ליצור ידית זו מקודם עם הגישה GENERIC_READ או GENERIC_WRITE לקובץ. הפרמטר lpFileSizeHigh מצביע למשתנה שבו מחזירה הפונקציה את מילת הסדר הגבוהה של גודל הקובץ. אם היישום אינו צריך את מילת הסדר הגבוהה, הפרמטר lpFileSizeHigh יכול להיות שווה ל- NULL.

אם הפונקציה מצליחה, הערך המוחזר הוא מילת הסדר הנמוך של המילה הכפולה שמציינת את גודל הקובץ; ואם lpFileSizeHigh אינו NULL, הפונקציה ממקמת את מילת הסדר הגבוהה של המילה הכפולה שמציינת את גודל הקובץ במשתנה שמוצבע על ידי הפרמטר. אם הפונקציה נכשלת והערך של lpFileSizeHigh הוא NULL, הפונקציה מחזירה 0xFFFFFFFF. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל- GetLastError. אם הפונקציה נכשלת ו-lpFileSizeHigh אינו NULL, הפונקציה מחזירה 0xFFFFFFFF, ו-GetLastError מחזירה ערך שונה מ-NO_ERROR.

אינך יכול להשתמש בפונקציה GetFileSize עם ידית של התקן שאינו תומך בחיפוש (Non-Seeking Device), כמו למשל צינור או התקן תקשורת. כדי לקבוע את סוג הקובץ עבור hFile, צריך להשתמש בפונקציה GetFileType. הפונקציה GetFileSize משיגה את גודל הקובץ שאינו דחוס. צריך להשתמש בפונקציה GetCompressedFileSize כדי להשיג את גודל הקובץ הדחוס. שים לב, כי אם הפונקציה מחזירה 0xFFFFFFFF ו-lpFileSizeHigh אינו NULL, היישום חייב לקרוא ל-GetLastError כדי לקבוע אם הפונקציה הצליחה או נכשלה.

התקליטור שמצורף לספר זה מכיל את התוכנית **Check_FileSize** (בתיקיה Books\59285). התוכנית משתמשת בפונקציה GetFileType כדי לבדוק את סוג ידית הקובץ. אם הקובץ נמצא בדיסק, התוכנית תציג את המספר הסידורי וגודל הקובץ בתיבת ההודעה. אם הקובץ אינו בדיסק, הפונקציה מודיעה למשתמש שהקובץ אינו נמצא בדיסק ויוצאת.

17.17 חותמת הזמן של קובץ

בוודאי למדת לבדוק את חותמת הזמן של קבצים על ידי שימוש בפונקציות זמן הריצה של C (C Run-Time Functions). ממשק Windows API גם כן מספק פונקציה שבאפשרות התוכניות להשתמש בה להשגת חותמת הזמן של הקובץ (File's Timestamp). הפונקציה GetFileTime משיגה את התאריך והזמן של יצירת הקובץ, את מועד הגישה האחרונה לקובץ, ומועד השינוי האחרון של הקובץ. את הפונקציה GetFileTime כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL GetFileTime(
  HANDLE hFile,                // identifies the file
  LPFILETIME lpCreationTime,   // address of creation time
  LPFILETIME lpLastAccessTime, // address of last access time
  LPFILETIME lpLastWriteTime   // address of last write time
);
```

הפרמטר hFile מציין את שם הקובץ שהפונקציה צריכה להשיג את הזמנים והתאריכים הקשורים בו. התהליך חייב ליצור תחילה ידית עם הגישה GENERIC_READ לקובץ. הפרמטר lpCreationTime מצביע למבנה מסוג FILETIME שמקבל את התאריך והזמן שהקובץ נוצר בהם. הפרמטר lpLastAccessTime מצביע למבנה מסוג FILETIME שמקבל את התאריך והזמן של הגישה האחרונה לקובץ. זמן הגישה האחרונה מכיל את המועד האחרון שהקובץ נכתב ל-, נקרא מ-, ובמקרה של קבצי הפעלה - הופעל. הפרמטר lpLastWriteTime מצביע למבנה מסוג FILETIME שמקבל את התאריך והזמן של כתיבת הקובץ לאחרונה. אם הקובץ אינו דורש מידע של שלושת הזמנים האלה, אפשר להציב NULL בפרמטר הפונקציה שאין צורך בו.

אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, הפונקציה מחזירה את הערך אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError.

מערכת הקבצים של Windows 9x ושל Windows NT תומכות בערכי זמן יצירת הקובץ, זמן גישה אחרונה, וכתיבה אחרונה בקובץ. ב-Windows 9x הדיוק של זמן הקובץ הוא 2 שניות (פירוש הדבר, הזמן ש-Windows רושמת בו את המאפיינים יהיה בתחום שתי שניות מזמן הגישה הממשי). דיוק זמן הקבצים במערכות קבצים אחרות, כמו אלו הקשורות לרשת, תלוי במערכת הקבצים הרחוקה. ההתקן הרחוק יכול אולי להגביל את דיוק הזמן.

בתקליטור המצורף לספר זה תמצא את התוכנית **Get_Time**, אשר משמשת ב- GetFileTime כדי להשיג את הזמן הנוכחי של הקובץ ואחר כך הופכת את הזמן הזה לזמן המקומי. התוכנית הופכת אחר כך את הזמן המקומי שבמבנה FILETIME לפורמט הזמן והתאריך שמוכרים ל-DOS. כדי לראות את השינוי, יש להשתמש בסייר Windows.

17.18 יצירת ספריות/תיקיות

במערכת DOS למדת להשתמש בפונקציות זמן הריצה של ספריית C (C Run-Time Library Functions) כדי ליצור ספריה או תיקיה. באופן דומה, באפשרות תוכניות Windows להשתמש בפונקציה CreateDirectory של Win32 API כדי ליצור ספריה חדשה. אם מערכת הקבצים הקיימת תומכת גם כן באבטחת קבצים וספריות, הפונקציה מספקת את מתאר האבטחה המוגדר גם לספריה החדשה. זכור שלכל תהליך יש ספריה נוכחית משלו ולכן, קריאות ל-CreateDirectory אינן צריכות להניח שהספריה הנוכחית של התהליך היא קבועה. את הפונקציה CreateDirectory כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL CreateDirectory(
  LPCTSTR lpPathName,    // pointer to a directory path string
  LPSECURITY_ATTRIBUTES lpSecurityAttributes
                         // pointer to a security descriptor
);
```

הפרמטר lpPathName מצביע למחרוזת המסתיימת ב-NULL, אשר מגדירה את נתיב הספריה שצריך ליצור. גודל ברירת המחדל הקיים לגבול מספר התווים שבשם הנתיב הוא MAX_PATH תווים. גבול זה מתייחס לדרך שבה הפונקציה CreateDirectory מפרשת נתיבים. תחת Windows NT אפשר לעבור גבול זה.

הפרמטר lpSecurityAttributes הוא מצביע למבנה מסוג SECURITY_ATTRIBUTES אשר קובע אם תהליכי בן יכולים לרשת את הידית המוחזרת. אם lpSecurityAttributes הוא NULL, תהליכי הבן אינם יכולים לרשת את הידית. תחת Windows NT, האיבר lpSecurityDescriptor של המבנה מגדיר מתאר אבטחה עבור הספריה החדשה. אם lpSecurityAttributes הוא NULL, הספריה מקבלת מתאר אבטחה של ברירת המחדל. מערכת הקבצים הקיימת חייבת לתמוך באבטחת קבצים וספריות, כדי שיהיה השפעה לפרמטר זה. תחת Windows 9x, מערכת ההפעלה מתעלמת מהאיבר lpSecurityDescriptor.

אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; ואם הפונקציה נכשלת, הפונקציה מחזירה את הערך אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא לפונקציה GetLastError.

התקליטור שמצורף לספר זה מכיל את התוכנית **Create_NewDir** (בתיקיה Books\59285). התוכנית משתמשת בפונקציה CreateDirectory כדי ליצור ספריה חדשה בשם "New Directory" בכונן :C של המחשב שהתוכנית מופעלת ממנו כרגע.

✍ **הערה:** יש מערכות ניהול קבצים, כמו זו של NT ובקיצור (NTFS), שתומכות בדחיסה של קבצים בודדים ושל ספריות. בכרכים (volumes) שמפורמטים למערכות קבצים כאלו, ספריה חדשה יורשת את תכונת הדחיסה של ספרית האב שלה. באפשרות היישום לקרוא ל-CreateFile עם הדגל FILE_FLAG_BACKUP_SEMANTICS כדי להשיג ידית לספריה. כדי לראות דוגמת קוד, התבונן ב- CreateFile.

17.19 מידע לגבי הספריה/תיקיה הנוכחית, או שינוי ספריה/תיקיה

תוכניות יוצרות ספריות/תיקיות חדשות במערכת הקבצים. עם זאת נמצא שלרוב הן מבקשות מידע אודות הספריה/תיקיה הנוכחית, ולא יוצרות חדשות. הפונקציה GetCurrentDirectory משיגה את הספריה הנוכחית של התהליך הנוכחי. את הפונקציה GetCurrentDirectory כותבים בתוכניות כמו שרואים בהגדרה שלהלן:

```
DWORD GetCurrentDirectory(
  DWORD nBufferLength,     // size, in characters, of
                           // directory buffer
  LPTSTR lpBuffer          // address of buffer for
                           // current directory
);
```

הפרמטר nBufferLength מגדיר את האורך, בתווים, של חוצץ מחרוזת הספריה הנוכחית. אורך החוצץ חייב להכיל מקום עבור תו NULL המסיים. הפרמטר lpBuffer מצביע לחוצץ המחרוזת של הספריה הנוכחית. מחרוזת זו, המסתיימת ב-NULL מגדירה את הנתיב המוחלט של הספריה הנוכחית. אם הפונקציה מצליחה, הערך המוחזר מגדיר את מספר התווים שנכתבו לחוצץ, ללא תו NULL המסיים; ואם הפונקציה נכשלת, הפונקציה מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError. אם החוצץ שמוצבע על ידי lpBuffer אינו גדול מספיק, הערך המוחזר מגדיר את גודל החוצץ הדרוש, כולל מספר הבתים שצריך עבור תו NULL מסיים.

באופן דומה, באפשרות התוכניות להשתמש בפונקציה SetCurrentDirectory כדי לשנות את הספריה הנוכחית של התהליך הנוכחי לספריה אחרת שמוגדרת על ידי המשתמש. כך כותבים את הפונקציה SetCurrentDirectory בתוכניות:

```
BOOL SetCurrentDirectory(LPCTSTR lpPathName);
```

הפרמטר lpPathName מצביע למחרוזות המסתיימת ב-NULL ומגדירה את הנתיב של הספריה החדשה. פרמטר זה יכול להיות נתיב יחסי, או נתיב מפורט שלם. בשני המקרים, מערכת ההפעלה מחשבת את הנתיב המפורט השלם של הספריה המוגדרת ומאחסנת אותו כנתיב הספריה הנוכחית. אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; ואם הפונקציה נכשלת, היא מחזירה את הערך אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל- GetLastError.

לכל תהליך יש ספריה/תיקיה נוכחית יחידה שמורכבת משני חלקים:

- **מזהה דיסק** (Disk Designator). זוהי אות הכונן ואחריה נקודתיים, או שם שרת ושם משותף (לדוגמה, \\Servername\Sharename).
- ספריה במזהה הדיסק.

התקליטור שמצורף לספר זה מכיל את התוכנית **Create_Set** (בתיקיה Books\59285). כאשר מהדרים ומריצים את התוכנית מתוך debug, היא מציגה את הספריה הנוכחית. לבסוף היא משנה את הנתיב אל הספריה הנוכחית לפי הפרמטר שהיא מקבלת. השמת הפרמטר מתבצעת באמצעות לחיצה בתפריט Project > Settings ובכרטיסיה debug יש לשים את שם הספריה החדש - Program Arguments.

17.20 השגת הספריות Windows ו-System

בסעיף קודם למדת כיצד יכולות התוכניות להשיג את שם הספריה/תיקיה הנוכחית של התהליך הנוכחי. ככל שהתוכניות הופכות למורכבות יותר, נדרש פעמים רבות מידע על מקום הספריה Windows ושל הספריה System הכפופה לה. כדי לקבל מידע זה, צריך להשתמש בפונקציה GetWindowsDirectory, אשר משיגה את נתיב הספריה Windows (ששמה עשוי להיות שונה מ-c:\windows או c:\winnt). הספריה Windows מכילה קבצים של יישומים מבוססי Windows, קבצי אתחול וקבצי עזרה. משתמשים בפונקציה GetWindowsDirectory כמו בהגדרה שלהלן:

```
UINT GetWindowsDirectory(
  LPTSTR lpBuffer,  // address of buffer for Windows directory
  UINT uSize          // size of directory buffer
);
```

הפרמטר lpBuffer מצביע לחוצץ שמקבל את המחרוזת שמסתיימת ב-NULL ומכילה את פירוט הנתיב. נתיב זה אינו מסתיים בלוכסן הפוך, אלא אם ספריית Windows היא ספריית השורש. לדוגמה, אם השם הוא windows בכונן C, שם הנתיב של ספריית Windows שהפונקציה מקבלת הוא c:\windows. אם Windows הותקנה בספריית השורש של כונן C, הנתיב שמתקבל הוא C:\. הפרמטר uSize מגדיר את הגודל המקסימלי, בתווים, של החוצץ שמוגדר על ידי הפרמטר lpBuffer. צריך לקבוע את הפרמטר uSize לפחות ל-MAX_PATH, כדי ליהיה די מקום בחוצץ עבור פירוט הנתיב.

אם הפונקציה מצליחה, הערך המוחזר הוא האורך, בתווים, של המחרוזת שהעתיקה הפונקציה לחוצץ, ללא תו NULL המסיים. אם האורך גדול מגודל החוצץ, הפונקציה מחזירה את גודל החוצץ הנדרש לה, כדי להחזיק בו את פירוט הנתיב; ואם הפונקציה נכשלת, הפונקציה מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError.

ספריית Windows היא הספריה שבה היישום צריך לאחסן קבצי אתחול ועזרה. אם המשתמש מריץ גירסה שיתופית של Windows, מערכת ההפעלה מוודאת שלכל משתמש יש ספריית Windows פרטית. אם יישום אשר יוצר קבצים אחרים רוצה לאחסן אותם בנפרד לכל משתמש (Per-User Basis), עליו למקם אותם בספריה שמוגדרת על ידי משתנה הסביבה HOMEPATH. המשתנה HOMEPATH תמיד מגדיר אחד משני דברים, את ספריית הבית אשר מובטח שהיא פרטית לכל משתמש, או את ספריית ברירת המחדל (לדוגמה, c:\users\default) אשר יש למשתמש בה את כל הגישות.

כמו שהתוכניות דורשות מידע על מקום ספריית Windows, הן גם צריכות לעיתים מידע אודות ספריית המערכת, System. הפונקציה GetSystemDirectory משיגה את נתיב ספריית המערכת של Windows. ספריית המערכת מכילה קבצים שונים, כמו רשימת ספריות Windows, מנהלי התקנים וקבצי גופנים. את הפונקציה GetSystemDirectory כותבים בתוכניות כמו בהגדרה שלהלן:

```
UINT GetSystemDirectory(
  LPTSTR lpBuffer,   // address of buffer for system directory
  UINT uSize         // size of directory buffer
);
```

הפרמטרים של הפונקציה GetSystemDirectory הם כמו אלה של הפונקציה GetWindowsDirectory. אם הפונקציה GetSystemDirectory מצליחה, הערך המוחזר הוא האורך, בתווים, של המחרוזת שהעתיקה הפונקציה לחוצץ, ללא תו NULL המסיים. אם האורך גדול מגודל החוצץ, הפונקציה מחזירה את גודל החוצץ הנדרש לה כדי להחזיק בו את פירוט הנתיב; אם הפונקציה נכשלת, היא מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל- GetLastError.

התקליטור שמצורף לספר זה מכיל את התוכנית **Show_Windows** (בתיקיה Books\59285). כאשר מהדרים ומריצים את התוכנית הזו, רואים שהיא משתמשת בפונקציות GetWindowsDirectory ו- GetSystemDirectory כדי להשיג את שמות הספריות Windows ו- System. התוכנית מציגה אחר כך את שמות הספריות בשטח הלקוח של החלון.

✍ **הערה:** ככלל, היישומים אינם צריכים ליצור קבצים בספריה System. אם המשתמש מריץ גירסה שיתופית של Windows, אין ליישום גישת כתיבה לספריה System. היישומים צריכים ליצור קבצים רק בספריה שמוחזרת על ידי הפונקציה GetWindowsDirectory.

17.21 העברת ספריות/תיקיות

כשם שתוכניות יכולות ליצור ספריות/תיקיות זמניות, או ספריות/תיקיות לשימוש פנימי, יכולים להיות מצבים שבהם התוכניות חייבות להעביר ספריות/תיקיות קיימות ממקומן. הפונקציה RemoveDirectory מוחקת ספריה/תיקיה ריקה קיימת.

את הפונקציה RemoveDirectory כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL RemoveDirectory(LPCTSTR lpPathName);
```

הפרמטר lpPathName מצביע למחרוזת המסתיימת ב-NULL שמגדירה את הנתיב של הספריה שצריך למחוק אותה. הנתיב חייב להגדיר ספריה ריקה, והתהליך הקורא חייב להיות בעל גישת מחיקה לספריה. אם הפונקציה מצליחה, הערך המוחזר הוא שונה מאפס. אם הפונקציה נכשלת, הפונקציה מחזירה את הערך אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל- GetLastError.

התקליטור שמצורף לספר זה מכיל את התוכנית **Create_Remove** (בתיקיה Books\59285). כאשר מהדרים ומריצים את התוכנית **Create_Remove**, התוכנית יוצרת ספריה חדשה בשם Child Folder, ומודיעה למשתמש על כך. אחר כך, התוכנית מוחקת את הספריה החדשה שיצרה, ומודיעה למשתמש על השינוי הזה גם כן.

17.22 העתקת קבצים

לרשות התוכניות עומדים כלים רבים של Windows לניהול קבצים. ככל שהתוכניות הולכות ונעשות יותר מורכבות, הן צריכות לעשות פעולות שונות בקבצים, וביניהם - להעתיק קובץ ממקום אחד למקום אחר, כאשר הכוונה לכך שקובץ המקור ישאר במקומו הנוכחי ורק העתק שלו יועבר למקום אחר. הפונקציה CopyFile מעתיקה קובץ קיים לקובץ חדש, ויכולה לקבוע לו שם חדש. כותבים את הפונקציה כמו בהגדרה שלהלן:

```
BOOL CopyFile(
  LPCTSTR lpExistingFileName,
                          // pointer to name of an existing file
  LPCTSTR lpNewFileName, // pointer to filename to copy to
  BOOL bFailIfExists     // flag for operation if file exist
);
```

הפרמטר lpExistingFileName מצביע למחרוזת המסתיימת ב-NULL אשר מגדירה שם של קובץ קיים. הפרמטר lpNewFileName מצביע למחרוזת המסתיימת ב-NULL אשר מגדירה את שם הקובץ החדש. הפרמטר bFailIfExists מגדיר איך הפונקציה ממשיכה, אם קיים כבר קובץ בעל שם שכבר מוגדר בפרמטר lpNewFileName. אם הפרמטר bFailIfExists הוא True ושם הקובץ החדש קיים כבר, הפונקציה נכשלת; אם פרמטר זה הוא False ושם הקובץ החדש קיים כבר, הפונקציה דורסת את הקובץ הקיים ומסיימת.

אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, היא מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError. מאפייני האבטחה של הקובץ הקיים אינם מועתקים לקובץ החדש.

תכונות הקובץ (*_FILE_ATTRIBUTE) של קובץ קיים מועתקות אל הקובץ החדש. לדוגמה, אם לקובץ קיים יש תכונה FILE_ATTRIBUTE_READONLY, העתקת הקובץ שנעשית על ידי קריאה לפונקציה CopyFile גורמת לכך שגם ההעתק מקבל את התכונה FILE_ATTRIBUTE_READONLY.

התקליטור שמצורף לספר זה מכיל את התוכנית **Create_Copy** (בתיקיה Books\59285). כאשר מהדרים ומריצים את התוכנית **Create_Copy**, היא יוצרת את הקובץ file1.txt. כאשר המשתמש בוחר באפשרות !Test, התוכנית מעתיקה את הקובץ file1.txt לקובץ file2.txt. אם הקובץ file2.txt קיים כבר, התוכנית מבקשת מהמשתמש אישור להחליף את הקובץ הקיים.

17.23 העברת קבצים ושינוי שם

תוכניות יכולות ליצור בקלות העתק של קובץ ולכתוב אותו במקום אחר. פעמים רבות התוכניות צריכות להעביר קובץ או ספריה/תיקיה למקום אחר, מבלי לשמור על ההעתק המקורי של הקובץ. הפונקציה MoveFile משנה שם של קובץ קיים או ספריה/תיקיה (כולל כל רמות המשנה שלה). את הפונקציה MoveFile כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL MoveFile(
  LPCTSTR lpExistingFileName,      // address of name of the
                                   // existing file
  LPCTSTR lpNewFileName            // address of new name for
                                   // the file
);
```

הפרמטר lpExistingFileName מצביע למחרוזת המסתיימת ב-NULL ואשר מגדירה שם של קובץ קיים או שם של ספריה/תיקיה קיימת. הפרמטר lpNewFileName מצביע למחרוזת המסתיימת ב-NULL, שמגדירה את שם הקובץ החדש, או את שם הספריה/התיקיה החדשה. אסור שהשם החדש יהיה קיים. את הקובץ החדש אפשר לכתוב בספריה שונה או בכונן אחר. ספריה חדשה חייבת להיות באותו כונן. אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, הפונקציה מחזירה את הערך אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל- GetLastError.

הפונקציה MoveFile מעבירה (משנה שם) קובץ או ספריה (כולל כל ספריות המשנה) באותה ספריה או לספריות אחרות. המגבלה היחידה שיש לפונקציה MoveFile היא בכך שהיא נכשלת בהעברת ספריות, כאשר היעד הוא **בכרך שונה** (Different Volume).

התקליטור שמצורף לספר זה מכיל את התוכנית **Move_Folder** (בתיקיה Books\59285). כשהמשתמש בוחר באפשרות !Test, התוכנית מעבירה אחת מהספריות ממבנה הספריות הקיים אל ספריה אחרת.

17.24 מחיקת קבצים בספריה/תיקיה

תוכניות מפעילות את הפונקציה RemoveDirectory כדי למחוק ספריה/תיקיה במערכת הקבצים. אך, הספריה חייבת להיות ריקה לפני הקריאה לפונקציה RemoveDirectory, או שהפונקציה נכשלת. כדי למחוק קבצים בספריה, התוכניות יכולה להשתמש בפונקציה DeleteFile. משתמשים בפונקציה DeleteFile בתוכניות כמו שרואים בהגדרה שלהלן:

```
BOOL DeleteFile(LPCTSTR lpFileName);
```

הפרמטר lpFileName מצביע למחרוזת המסתיימת ב-NULL, ומגדיר את הקובץ שהפונקציה צריכה למחוק. אם הפונקציה מצליחה, הערך המוחזר הוא שונה מאפס; אם הפונקציה נכשלת, היא מחזירה את הערך אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError.

אם יישום מנסה למחוק קובץ שאינו קיים, הפונקציה DeleteFile נכשלת. תחת Windows 9x, הפונקציה DeleteFile מוחקת קובץ, אפילו אם הוא פתוח כרגע לקלט/פלט, או שהוא פתוח כמיפוי קובץ בזיכרון. כדי למנוע שגיאות, צריך לסגור את הקבצים לפני שמנסים למחוק אותם. תחת Windows NT, הפונקציה DeleteFile נכשלת אם יישום מנסה למחוק קובץ שפתוח כרגע לקלט/פלט או שהוא פתוח כמיפוי קובץ בזיכרון.

17.25 הפונקציה FindFirstFile לאיתור קבצים

בוודאי מוכרות לך הדרכים לחפש קובץ בספריה/תיקיה נוכחית באמצעות פונקציות זמן הריצה של C (C Run-Time Functions). בתוכניות Windows, צריך להשתמש ב**פונקציות החיפוש** (Find Functions) כדי לאתר קבצים שעומדים בקריטריון רצוי. שתי פונקציות חיפוש נותנות את השירות הזה לתוכניות: הפונקציה FindFirstFile מחפשת בספריה קובץ אשר השם שלו תואם את שם הקובץ המוגדר; והפונקציה FindFirstFileEx בוחנת גם שמות ספריות משנה (Subdirectory) וגם שמות קבצים. את הפונקציה FindFirstFile כותבים בתוכניות כמו בהגדרה שלהלן:

```
HANDLE FindFirstFile(
  LPCTSTR lpFileName,               // pointer to name of the
                                    // file to search for
  LPWIN32_FIND_DATA lpFindFileData  // pointer to
);                                  // returned information
```

במערכות Windows 9x ו-Windows NT, הפרמטר lpFileName מצביע למחרוזת המסתיימת ב-NULL ואשר מגדירה ספריה תקפה או נתיב ושם קובץ, אשר יכולים לכלול תווי הכללה (* ו- ?). עם זאת, תחת Windows 9x, מחרוזת זו חייבת להיות קטנה מ-MAX_PATH תווים.

הפרמטר lpFindFileData מצביע למבנה WIN32_FIND_DATA שמקבל מידע על הקובץ שנמצא או ספריית המשנה. באפשרות התוכניות להשתמש במבנה זה בקריאות עוקבות לפונקציות FindNextFile או FindClose כדי להתייחס לקובץ או לספריית המשנה. המבנה WIN32_FIND_DATA מתאר קובץ אשר נמצא על ידי אחת הפונקציות FindFirstFile, FindFirstFileEx או FindNextFile. המבנה WIN32_FIND_DATA מוגדר ב-Win32 API כפי שמוצג להלן:

```
typedef struct _WIN32_FIND_DATA {
  DWORD dwFileAttributes;
  FILETIME ftCreationTime;
  FILETIME ftLastAccessTime;
  FILETIME ftLastWriteTime;
  DWORD nFileSizeHigh;
  DWORD nFileSizeLow;
  DWORD dwReserved0;
  DWORD dwReserved1;
  TCHAR cFileName[ MAX_PATH ];
  TCHAR cAlternateFileName[ 14 ];
} WIN32_FIND_DATA;
```

טבלה 17.17 מסבירה בפירוט את המבנה WIN32_FIND_DATA.

טבלה 17.17: מרכיבי המבנה WIN32_FIND_DATA.

פרמטר	תיאור
dwFileAttributes	מגדיר את תכונות הקובץ שנמצא. איבר זה יכול להיות ערך אחד או יותר מהערכים שמפורטים בטבלה 17.16.
ftCreationTime	מגדיר מבנה FILETIME אשר מכיל את הזמן שבו נוצר הקובץ. FindFirstFile ו- FindNextFile מדווחות על זמני קובץ בפורמט זמן עולמי (Coordinated Time Format ,UTC). פונקציות אלו קובעות את האיברים של FILETIME לאפס, אם מערכת הקבצים אשר מכילה את הקובץ אינה תומכת באיבר מסוג זה. באפשרותך להשתמש בפונקציה FileTimeToLocalFileTime כדי להפוך את יחידות הזמןm-UTC לזמן מקומי, ואחר כך להשתמש בפונקציה FileTimeToSystemTime כדי להפוך את הזמן המקומי למבנה SYSTEMTIME אשר מכיל איברים נפרדים של **חודש** (Month), **יום** (Day), **שנה** (Year), **יום בשבוע** (Weekday), **שעה** (Hour), **דקה** (Minute), **שנייה** (Second), ו- **מילישנייה** (Millisecond).

פרמטר	תיאור
ftLastAccessTime	מגדיר מבנה FILETIME אשר מכיל את זמן הגישה האחרונה לקובץ. זמן זה הוא בפורמט UTC ; האיברים של FILETIME הם אפס, אם מערכת הקבצים אינה תומכת באיבר זמן זה.
ftLastWriteTime	מגדיר מבנה FILETIME אשר מכיל את הזמן של הפעם האחרונה שהקובץ נכתב. זמן זה הוא בפורמט UTC ; איברי FILETIME הם אפס אם מערכת הקבצים אינה תומכת באיבר זמן זה.
nFileSizeHigh	מגדיר את ערך מילת הסדר הגבוהה DWORD המייצג את גודל הקובץ בבתים. ערך זה הוא אפס, אלא אם גודל הקובץ גדול מ-MAXDWORD. גודל הקובץ שווה ל- (nFileSizeHigh * MAXDWORD) + nFileSizeLow.
nFileSizeLow	מגדיר את ערך מילת הסדר הנמוך DWORD המייצג את גודל הקובץ בבתים.
dwReserved0	שמור לשימוש עתידי.
dwReserved1	שמור לשימוש עתידי.
cFileName	מחרוזת המסתיימת ב-NULL שהיא למעשה שם הקובץ.
cAlternateFileName	מחרוזת המסתיימת ב-NULL שהיא שם חלופי לקובץ. שם זה הוא בפורמט הקלסי : 8.3 (Filename.ext).

אם שם הקובץ הוא בפורמט ארוך, השם המלא מוצג בשדה cFileName וגירסת פורמט 8.3 המצומצמת מוצגת בשדה cAlternateFileName ; אחרת, cAlternateFileName ריק. לחילופין, תוכל להשתמש בפונקציה GetShortPathName כדי למצוא את גירסת השם המצומצמת בפורמט 8.3.

אם הפונקציה מצליחה, היא מחזירה ידית חיפוש שמשתמשים בה בקריאות עוקבות ל-FindNextFile או FindClose ; אם הפונקציה נכשלת, היא מחזירה INVALID_HANDLE_VALUE. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError.

הפונקציה FindFirstFile פותחת ידית חיפוש ומחזירה מידע על הקובץ הראשון אשר שמו חופף לתבנית המוגדרת. לאחר שידית החיפוש נוצרת, תוכל להשתמש בפונקציה FindNextFile כדי לחפש קבצים אחרים אשר תואמים לאותה תבנית. כאשר אין יותר צורך בידית החיפוש, צריך להשתמש בפונקציה FindClose כדי לסגור את הידית. פונקציה זו מחפשת קבצים רק לפי שם ; אינך יכול להשתמש בה לחיפושים מבוססי תכונות (Attribute-Based Searches).

17.26 הפונקציה FindNextFile

בסעיף קודם למדת על הפונקציה FindFirstFile, אשר מוצאת את המופע הראשון של קובץ בעץ ספריות, אשר תואם לשם קובץ נתון. כדי להמשיך בחיפוש קבצים נוספים בעלי אותו שם (או תואמים לתווי הכללה), התוכניות חייבות להשתמש בפונקציית חיפוש נוספת. הפונקציה FindNextFile ממשיכה בחיפוש הקבצים מהקריאה הקודמת של הפונקציה FindFirstFile. את הפונקציה FindNextFile כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL FindNextFile(
  HANDLE hFindFile,                   // handle to search
  LPWIN32_FIND_DATA lpFindFileData    // pointer to structure
                                      // for found file
);
```

הפרמטר hFindFile מזהה ידית חיפוש שהוחזרה על ידי קריאה קודמת אל הפונקציה FindFirstFile. הפרמטר lpFindFileData מצביע למבנה מסוג WIN32_FIND_DATA אשר מקבל מידע על הקובץ שנמצא, או על ספריית המשנה. באפשרותך להשתמש במבנה זה עבור קריאות עוקבות ל-FindNextFile ולהתייחס לקובץ או לספריה הרצויים.

אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, היא מחזירה את הערך אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל- GetLastError. אם GetLastError אינה מוצאת קבצים תואמים, היא מחזירה ERROR_NO_MORE_FILES. הפונקציה FindNextFile מחפשת קבצים רק לפי שם; אינך יכול להשתמש בה לחיפושים מבוססי תכונות (Attribute-Based Searches). כדי ללמוד על חיפושים מסוג זה, קרא בסעיף 17.28.

בתקליטור שמצורף לספר זה תמצא את התוכנית **Walk_Directories** (בתיקיה Books\59285). התוכנית **Walk_Directories** משתמשת בשתי הפונקציות FindFirstFile ו-FindNextFile לחיפוש בכונן הדיסקים שאתה מגדיר. בנוסף, התוכנית משתמשת בפונקציה רקורסיבית כדי להיות בטוחה שהחיפוש נעשה בכל הספריות שבדיסק הנוכחי. ככל שהתוכנית מחפשת, היא מוסיפה שמות קבצים לתיבת הרשימה המוצגת.

17.27 סגירת ידית החיפוש עם FindClose

למדת שכאשר התוכניות מפעילות את הפונקציות FindFirstFile ו-FindNextFile, הן פותחות ידית שונה (ידית חיפוש) עבור פונקציות אלו מזו שמערכת ההפעלה מחזירה באופן נורמלי על ידי הקריאה ל-CreateFile. אם תנסה לסגור ידית חיפוש עם CloseHandle, תקבל מצב שגיאה. על כן, התוכניות חייבות להשתמש בפונקציה FindClose כדי לסגור את ידית החיפוש. את הפונקציה FindClose כותבים כמו בהגדרה שלהלן:

```
BOOL FindClose(HANDLE hFindFile);
```

הפרמטר hFindFile מזהה את ידית החיפוש. התוכנית חייבת לפתוח תחילה את ידית החיפוש על ידי קריאה לפונקציה FindFirstFile. אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, היא מחזירה אפס. אחרי שהתוכנית קוראת לפונקציה FindClose, היא אינה יכולה להשתמש בידית החיפוש, שמוגדרת על ידי הפרמטר hFindFile, בקריאות עוקבות לפונקציה FindNextFile או בקריאות לפונקציה FindClose. תוכל לראות שהתוכנית **Walk_Directories**, המפורטת בסעיף 17.26, משתמשת בפונקציה FindClose רק כאשר המשתמש מסיים לחפש בכל ספריות הקבצים.

17.28 חיפוש לפי תכונות עם פונקציות חיפוש קבצים

כמו שלמדת בסעיף קודם, התוכניות משתמשות בפונקציית חיפוש רקורסיבית (WalkDirsRecurse בתוכנית **Walk_Directories**) כדי לחפש קובץ רצוי בכל הכונן. אם אתה מתכנן יישום שרק המשתמשים ב- Windows NT 4.0 ישתמשו בו, תוכל להשתמש בפונקציה FindFirstFileEx (יחד עם הפונקציה FindNextFile) כדי לחפש בספריה/תיקיה רצויה. הפונקציה FindFirstFileEx מחפשת בספריה מוגדרת את הקובץ שהשם והתכונות שלו חופפים לאלה שאתה מגדיר בקריאה לפונקציה. את הפונקציה FindFirstFileEx עליך לכתוב כמו בהגדרה שלהלן:

```
HANDLE FindFirstFileEx(
  LPCTSTR lpFileName,           // pointer to the file's
                                // name to search for
  FINDEX_INFO_LEVELS fInfoLevelId, // information level of
                                // the returned data
  LPVOID lpFindFileData,        // pointer to the returned
                                // information
  FINDEX_SEARCH_OPS fSearchOp,  // type of filtering to
                                // perform
  LPVOID lpSearchFilter,        // pointer to search criteria
  DWORD dwAdditionalFlags       // additional search
                                // control flags
);
```

הפרמטר lpFileName מצביע למחרוזת המסתיימת ב-NULL ומגדירה ספריה/תיקיה תקפה או נתיב ושם קובץ, כמו בפונקציה FindFirstFile, אלא שהפרמטר lpFileName יכול להכיל תווי הכללה (* ו- ?). הפרמטר fInfoLevelId מציג את פירוט סוגי FINDEX_INFO_LEVELS אשר נותנים את רמת המידע המוחזר. הפרמטר lpFindFileData מחזיר מצביע לנתוני הקובץ. רמת המידע שמוגדרת בפרמטר fInfoLevelId קובעת את סוג המצביע. הפרמטר fSearchOp מפרט את סוגי FINDEX_SEARCH_OPS אשר מודיעים לפונקציה על סוג הסינון שצריך לבצע מעבר להתאמת תווי הכללה. אם

פירוט סוגי fSearchOp כולל מידע חיפוש מבני, lpSearchFilter מצביע לקריטריון החיפוש. עד עכשיו, אף לא ערך אחד מהערכים הנתמכים על ידי הפרמטר fSearchOp דרש מידע חיפוש מורחב (למרות שגרסאות עתידיות של Windows NT אולי ידרשו מידע כזה). לכן, מצביע זה חייב להיות NULL. הפרמטר dwAdditionalFlags מגדיר דגלים נוספים לבקרת החיפוש. באפשרותך להשתמש בדגל FIND_FIRST_EX_CASE_SENSITIVE עבור חיפושים תלויי רישיות (Case-Sensitive). חיפוש ברירת המחדל אינו חיפוש תלוי רישיות. מערכת Windows NT 4.0 אינה מגדירה כל דגל אחר, אך גרסאות עתידיות של Windows NT אולי יגדירו דגלים נוספים.

אם הפונקציה מצליחה, היא מחזירה ידית חיפוש שהתוכניות יכולות להשתמש בה בקריאות עוקבות לפונקציות FindNextFile או FindClose; אם הפונקציה נכשלת, היא מחזירה INVALID_HANDLE_VALUE. הפונקציה FindFirstFileEx מאפשרת לפתוח ידית חיפוש ולהחזיר מידע אודות הקובץ הראשון ששמו תואם לתבנית ולתכונות שהוגדרו בקריאה לפונקציה. אם מערכת הקבצים הקיימת אינה תומכת בסוג הסינון שמוגדר על ידי הפרמטר fSearchOp, ששונה מסינון ספריה, FindFirstFileEx נכשלת והפונקציה מחזירה את קוד השגיאה ERROR_NOT_SUPPORTED. התוכנית חייבת במקרה זה להשתמש בפונקציה FileExSearchNameMatch לביצוע הסינון. לקבלת מידע נוסף אודות הפונקציה FileExSearchNameMatch, פנה לתיעוד הנלווה למהדר שלך.

כשאתה מיישם את ידית החיפוש, יכולה התוכנית שלך להשתמש בידית זו בפונקציה FindNextFile, כדי לחפש קבצים אחרים שתואמים את אותו מבנה סינון. כאשר התוכנית אינה צריכה יותר את ידית החיפוש, היא צריכה לסגור אותה על ידי הפונקציה FindClose.

✍ **הערה:** בערכת הפיתוח SDK של Windows NT 4.0 ניתן הסבר מפורט אודות סוגי **FindFirstFileEx**.

17.29 חיפוש באמצעות SearchPath ולא על ידי Find

למדת שהתוכניות מפעילות את פונקציות Find כדי לחפש קבצים בספריה/תיקיה. לחילופין, התוכניות יכולות להשתמש גם בפונקציה SearchPath כדי לחפש את הקובץ הרצוי. עם זאת, הפונקציה SearchPath מחפשת את הקובץ רק בקבוצת נתיבים מסוימת, כמפורט בטבלה 17.18. את הפונקציה SearchPath כותבים כמו בהגדרה שלהלן:

```
DWORD SearchPath(
  LPCTSTR lpPath,        // address of search path
  LPCTSTR lpFileName,    // address of filename
  LPCTSTR lpExtension,   // address of extension
  DWORD nBufferLength,   // size, in characters, of buffer
  LPTSTR lpBuffer,       // address of buffer for found filename
  LPTSTR *lpFilePart     // address of pointer to file component
);
```

הפונקציה SearchPath מקבלת את הפרמטר שמפורטים בטבלה 17.18.

טבלה 17.18: הפרמטרים שמקבלת הפונקציה **SearchPath**.

פרמטר	תיאור
lpPath	מצביע למחרוזת המסתיימת ב-NULL ומגדירה את הנתיב שהפונקציה מחפשת בו את הקובץ. אם פרמטר זה הוא NULL, הפונקציה מחפשת קובץ תואם בספריות/תיקיות העוקבות לפי הסדר שלהלן: 1. הספריה שהיישום נטען ממנה. 2. הספריה הנוכחית. 3. תחת Windows 9x - בספריית המערכת של Windows. השתמש בפונקציה GetSystemDirectory כדי לקבל את הנתיב של ספריה זו. תחת Windows NT - בספריית 32 סיביות של המערכת. השתמש בפונקציה GetSystemDirectory כדי לקבל את הנתיב של ספריה זו. השם של ספרית 32 סיביות של המערכת הוא בדרך כלל SYSTEM32. 4. תחת Windows NT - ספריית 16 סיביות של המערכת. אין פונקציה של Win32 שמקבלת את הנתיב של הספריה הזו, אך הפונקציה מחפשת אותה. בדרך כלל, שם ספריה זו הוא SYSTEM. 5. ספריית Windows. השתמש בפונקציה GetWindowsDirectory כדי לקבל את נתיב הספריה הזו. 6. הספריות שנמצאות במשתנה הסביבה PATH של Windows.
lpFileName	מצביע למחרוזת המסתיימת ב-NULL ומגדירה את שם הקובץ שצריך לחפש.
lpExtension	מצביע למחרוזת המסתיימת ב-NULL ומגדירה סיומת שהפונקציה מוסיפה לשם הקובץ כאשר היא מחפשת את הקובץ. התו הראשון של סיומת שם הקובץ חייב להיות נקודה. הפונקציה מוסיפה את הסיומת רק אם שם הקובץ שאתה מגדיר אינו כולל סיומת. אם התוכנית אינה דורשת סיומת שם קובץ, או אם שם הקובץ מכיל סיומת, פרמטר זה יכול להיות NULL.
nBufferLength	האורך בתווים של החוצץ שמקבל את הנתיב התקף ואת שם הקובץ.
lpBuffer	מצביע לחוצץ שמכיל את הנתיב התקף ושם הקובץ של הקובץ שנמצא.
lpFilePart	מצביע לכתובת (שב-lpBuffer) של הרכיב האחרון של הנתיב התקף ושם הקובץ. זו כתובת התו שלאחר הלוכסן ההפוך האחרון (\) בפירוט הנתיב.

אם הפונקציה SearchPath מצליחה, היא מחזירה את האורך, בתווים, של המחרוזת שהיא העתיקה לחוצץ, ללא תו NULL המסיים. אם הערך המוחזר (כלומר, אורך המחרוזת) גדול מ-nBufferLength, הערך שהתונקציה SearchPath מחזירה הוא גודל החוצץ שדרוש לה בכדי להחזיק את פירוט הנתיב. אם הפונקציה נכשלת, הפונקציה מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError.

התקליטור שמצורף לספר זה מכיל את התוכנית **Search_For_Calc** (בתיקיה Books\59285). כאשר מהדרים ומריצים את התוכנית **Search_For_Calc**, היא משתמשת בפונקציה SearchPath כדי לחפש בכונן את הקובץ calc.exe. אם התוכנית מוצאת את הקובץ calc.exe, היא מציגה את נתיב הקובץ בתיבת הודעה.

17.30 קבלת נתיב זמני

בתוכניות C למדת בוודאי ליצור קובץ זמני, ואתה יודע שאחד הצעדים שצריך לעשות לשם כך הוא לקבוע אם הנתיב הזמני נלקח ממשתנה הסביבה TEMP או TMP. ב-Windows, תוכל להשתמש בפונקציה GetTempPath כדי להשיג את הנתיב של הספריה/תיקיה ש-Windows משתמשת בה לקבצים זמניים. את הפונקציה GetTempPath כותבים בתוכנית כמו בהגדרה שלהלן:

```
DWORD GetTempPath(
  DWORD nBufferLength, // size, in characters, of the buffer
  LPTSTR lpBuffer       // address of buffer for temp. path
);
```

הפרמטר nBufferLength מגדיר את הגודל, בתווים, של חוצץ המחרוזת מפורט בפרמטר lpBuffer. הפרמטר lpBuffer מצביע לחוצץ המחרוזת, שמקבל את המחרוזת שמסתיימת ב-NULL ומגדירה את נתיב הקובץ הזמני.

אם הפונקציה GetTempPath מצליחה, היא מחזירה את האורך, בתווים, של המחרוזת שהתוכנית העתיקה לפרמטר lpBuffer, לא כולל את תו NULL המסיים. אם הערך המוחזר (למעשה, מחרוזת הנתיב) גדול מ-nBufferLength, הערך המוחזר הוא הגודל של החוצץ שדרוש לפונקציה בכדי להחזיק את מחרוזת הנתיב בשלמותה. אם הפונקציה נכשלת, הפונקציה מחזירה אפס.

הפונקציה GetTempPath מקבלת את הנתיב הזמני בצורה זו:

1. הנתיב שמוגדר על ידי משתנה הסביבה TMP.
2. הנתיב שמוגדר על ידי משתנה הסביבה TEMP, אם Windows אינה מגדירה TMP.
3. את הספריה הנוכחית, אם Windows אינה מגדירה TMP או TEMP.

17.31 יצירת קבצים זמניים

קל ופשוט ליצור קבצים זמניים בתוכניות Windows. הפונקציה GetTempFileName יוצרת שם עבור קובץ זמני. שם הקובץ הוא השרשור של נתיב ומחרוזת שאתה מגדיר כקידומת, תבנית של מחרוזת הקסדיצמלית ממספר שלם שאתה מגדיר, והסיומת TMP. המספר השלם שאתה מגדיר יכול להיות שונה מאפס. במקרה כזה, הפונקציה GetTempFileName יוצרת את שם הקובץ, אך אינה יוצרת את הקובץ עצמו. אם אתה מגדיר "אפס" עבור המספר השלם, הפונקציה יוצרת שם ייחודי ויוצרת את הקובץ בספריה המוגדרת. את הפונקציה GetTempFileName כותבים כמו בהגדרה שלהלן:

```
UINT GetTempFileName(
  LPCTSTR lpPathName,           // address of directory name
                                // for temporary file
  LPCTSTR lpPrefixString,       // address of filename prefix
  UINT uUnique,                 // number used to create
                                // temporary filename
  LPTSTR lpTempFileName         // address of buffer that
                                // receives the new filename
);
```

הפרמטר lpPathName מצביע למחרוזת המסתיימת ב-NULL ומגדירה את נתיב הספריות/התיקיות של שם הקובץ. המחרוזת שמסתיימת ב-NULL חייבת להיות מורכבת מתווי ANSI. היישומים מציינים בדרך כלל נקודה או את תוצאת הפונקציה GetTempPath עבור הפרמטר lpPathName. אם פרמטר זה הוא NULL, הפונקציה נכשלת. הפרמטר lpPrefixString מצביע לקידומת מחרוזת המסתיימת ב-NULL. הפונקציה GetTempPath משתמשת בשלושת התווים הראשונים של קידומת המחרוזת המסתיימת ב-NULL כקידומת שם הקובץ. מחרוזת זו חייבת להיות מורכבת מתווי ANSI.

הפרמטר uUnique מגדיר מספר שלם לא מסומן, אשר הפונקציה הופכת אותו למחרוזת הקסדצימלית לשימוש ביצירת שם הקובץ הזמני. אם uUnique שונה מאפס, הפונקציה מוסיפה את המחרוזת ההקסדצימלית ל-lpPrefixString כדי ליצור את שם הקובץ הזמני. אם uUnique שונה מאפס, הפונקציה אינה יוצרת את הקובץ המוגדר, ואינה בוחנת אם שם הקובץ הוא ייחודי. אם uUnique שווה לאפס, הפונקציה GetTempPath משתמשת במחרוזת הקסדצימלית שגוזרת אותה מזמן המערכת הנוכחי. אם uUnique שווה לאפס, הפונקציה משתמשת בערכים שונים, עד שהיא מוצאת שם קובץ ייחודי, ואז היא יוצרת את הקובץ בספריה lpPathName.

הפרמטר lpTempFileName מצביע לחוצץ אשר מקבל את שם הקובץ הזמני. חוצץ זה הוא מחרוזת המסתיימת ב-NULL ומורכב מתווי ANSI. חוצץ זה צריך להיות לפחות באורך המוגדר על ידי המערכת בקבוע MAX_PATH (בדרך כלל, 255) כדי להכיל את כל מחרוזת הנתיב.

אם הפונקציה GetTempPath מצליחה, היא מחזירה את הערך המספרי הייחודי שמשתמשים בו בשם הקובץ הזמני. אם הפרמטר uUnique שונה מאפס, הפונקציה מחזירה את אותו מספר, כמו שהוסבר; אם הפונקציה נכשלת, היא מחזירה אפס.

הפונקציה GetTempFileName יוצרת שם קובץ זמני בפורמט path\preuuuu.TMP, כאשר path מייצג את הנתיב שמוגדר על ידי הפרמטר lpPathName, pre מייצג את שלוש האותיות הראשונות של המחרוזת lpPrefixString, ו-uuuu מייצג את הערך ההקסדצימלי שמוגדר בפרמטר uUnique. כאשר יוצאים מ-Windows, Windows אינה מוחקת בצורה אוטומטית את הקבצים הזמניים ששמותיהם נוצרו על ידי הפונקציה GetTempFileName.

אם הפרמטר uUnique שווה לאפס, GetTempFileName מנסה ליצור מספר ייחודי שמבוסס על זמן המערכת הנוכחי. אם קובץ עם שם הקובץ שנוצר קיים, GetTempFileName מגדילה את המספר באחד וחוזרת על הבדיקה עבור שם הקובץ הקיים. הפונקציה ממשיכה בבדיקות, עד שמוצאת שם קובץ ייחודי, ואז היא יוצרת קובץ עם השם הייחודי הזה וסוגרת אותו. כאשר uUnique שונה מאפס, הפונקציה GetTempFileName אינה מנסה ליצור ולפתוח את הקובץ.

התקליטור שמצורף לספר זה מכיל את התוכנית **Create_Temp**, אשר משתמשת בפונקציות GetTempPath ו- GetTempFileName כדי להשיג שם קובץ זמני. כאשר התוכנית מתחילה, היא יוצרת קובץ זמני לשימוש התוכנית ומציגה תיבת הודעה עם שם הקובץ הזמני. כאשר התוכנית מסתיימת, היא סוגרת ומוחקת את הקובץ.

17.32 הפונקציה CreateNamedPipe

למדת שהתוכניות יכולות להשתמש בצינורות (שמשתמשים בהם באופן כללי לתקשורת בין שני מחשבים או יותר) בצורה דומה לפעולות קלט ופלט של קבצים. כשרוצים להשתמש בתוכנית בצינור, חייבים ליצור אותו תחילה. הפונקציה CreateNamedPipe יוצרת מופע של צינור בעל שם (Named Pipe) בשרת צינור (Pipe Server) ומחזירה ידית לפעולות עוקבות בצינור. תהליך של שרת צינור בעל שם משתמש בפונקציה CreateNamedPipe לשני דברים: האחד - ליצור את המופע הראשון של צינור בעל שם שאתה מגדיר ולקבוע את התכונות הבסיסיות שלו; והשני - ליצור מופע נוסף של צינור בעל שם קיים. שים לב שבאפשרותך להשתמש בצינורות בעלי שם רק בתקשורת ברשתות.

את הפונקציה CreateNamedPipe כותבים בתוכניות כמו בהגדרה שלהלן:

```
HANDLE CreateNamedPipe(
  LPCTSTR lpName,              // pointer to pipe name
  DWORD dwOpenMode,            // pipe open mode
  DWORD dwPipeMode,            // pipe-specific modes
  DWORD nMaxInstances,         // maximum number of instances
  DWORD nOutBufferSize,        // output buffer size, in bytes
```

```
  DWORD nInBufferSize,         // input buffer size, in bytes
  DWORD nDefaultTimeOut,       // time-out time, in milliseconds
  LPSECURITY_ATTRIBUTES lpSecurityAttributes
                               // pointer to security attributes
);
```

הפונקציה CreateNamedPipe מקבלת את הפרמטרים שמפורטים בטבלה 17.19.

טבלה 17.19: הפרמטרים שמקבלת הפונקציה **CreateNamedPipe**.

פרמטר	תיאור
lpName	מצביע למחרוזת המסתיימת ב-NULL ומזהה את הצינור באופן ייחודי. המחרוזת חייבת להיות בפורמט \\.\pipe\pipename, רכיב pipename של השם יכול להכיל כל תו מלבד לוכסן הפוך, כולל מספרים ותווים מיוחדים. כל מחרוזת של שם צינור יכולה להגיע לאורך של 256 תווים. שמות הצינורות אינם תלויות רישיות (case sensitive).
DwOpenMode	מציין את מצב הגישה של הצינור, **מצב החפיפה** (Overlapped Mode), מצב **כתיבה כוללת** (Write-Through), **מצב גישה מאובטחת** (Security Access Mode) של ידית הצינור. הפרמטר dwOpenMode חייב להגדיר דגל אחד מדגלי מצב הגישה לצינור שמפורטים בטבלה 17.20 וחייב להגדיר אותו דגל מצב עבור כל מופע של הצינור.
DwPipeMode	מגדיר את מצבי הסוגים, הקריאה וההמתנה של ידית הצינור. הפרמטר dwPipeMode חייב להגדיר דגל אחד או יותר מדגלי מצב סוג הצינור שמפורטים בטבלה 17.21 ועליך להגדיר אותו מצב או מצבי סוג עבור כל מופע של הצינור. אם אתה מגדיר אפס, פרמטר ערך ברירת המחדל הוא מצב סוג-בית (Byte-Type Mode).
nMaxInstances	מגדיר את המספר המקסימלי של המופעים שהפונקציה יכולה ליצור עבר הצינור. הפרמטר nMaxInstances חייב להגדיר את אותו מספר עבור כל המופעים. ערכים מקובלים הם בתחום 1 עד PIPE_UNLIMITED_INSTANCES. אם פרמטר זה שווה ל- PIPE_UNLIMITED_INSTANCES, רק זמינות משאבי המערכת יכולה לגרום להגבלת מספר מופעי הצינור ש- CreateNamedPipe יכולה ליצור.
nOutBufferSize	מגדיר את מספר הבתים שצריך לשמור עבור חוצץ הפלט.
nInBufferSize	מגדיר את מספר הבתים שצריך לשמור עבור חוצץ הקלט.

פרמטר	תיאור
nDefaultTimeOut	מגדיר את ערך פסק הזמן של ברירת המחדל, במילישניות, אם הפונקציה WaiteNamedPipe מגדירה את NMPWAIT_USE_DEFAULT_WAIT. כל מופע של צינור בעל שם חייב להגדיר את אותו ערך.
lpSecurityAttributes	מצביע למבנה מסוג SECURITY_ATTRIBUTES אשר מגדיר **מתאר אבטחה** (Security Descriptor) עבור הצינור בעל השם וקובע אם תהליכי בן יכולים לרשת את הידית המוחזרת. אם lpSecurityAttributes שווה ל-NULL, הצינור בעל השם מקבל אתברירת המחדל של מתאר האבטחה ותהליכי בן אינם יכולים לרשת את הידית.

כמו שלמדת בטבלה 17.19, יש מספר ערכים של קבועים מובנים אשר Windows מאפשרת לך להשתמש בהם עבור הפרמטר dwOpenMode. טבלה 17.20 מפרטת את הערכים האפשריים של מצבים אלה.

טבלה 17.20: הערכים האפשריים עבור הפרמטר **dwOpenMode**.

מצב	תיאור
FILE_FLAG_WRITE_THROUGH	מאפשר מצב **כתיבה כוללת** (Write-Through). מצב זה משפיע רק על פעולות כתיבה בצינורות מסוג בית (Byte-Type), ואחר כך, רק כאשר תהליכי הלקוח והשרת נמצאים במחשבים שונים. אם FILE_FLAG_WRITE_THROUGH מאפשר מצב זה, פונקציות שכותבות לצינור בעל שם אינן חוזרות עד שהמערכת משדרת את הנתונים שנכתבו דרך הרשת ומעבירה אותם לחוצץ הצינור במחשב הרחוק. אם הקריאה ל-CreateNamedPipe אינה מאפשרת מצב זה, המערכת משפרת את יעילות הפעולות ברשת על ידי העברת הנתונים לחוצץ, עד שמספר מינימלי של בתים מצטבר, או עד שעובר זמן מקסימלי.
FILE_FLAG_OVERLAPPED	מאפשר **מצב חפיפה** (Overlapped Mode). אם הפונקציה CreateNamedPipe מאפשרת מצב זה, פונקציות שמבצעות קריאה, כתיבה, וחיברו (Connect), ועשויות לפעול זמן רב, יכולות לחזור מיד (והפעולה הרצויה תימשך בחפיפה). לדוגמה, במצב חפיפה, מטלה יכולה לטפל בפעולות קלט ופלט בו-זמנית במופעים רבים של צינור, או לבצע פעולות קריאה וכתיבה באותה ידית של הצינור.

מצב	תיאור
	אם הפונקציה CreateNamedPipe אינה מאפשרת מצב חפיפה, פונקציות שמבצעות פעולות קריאה, כתיבה, וחיבור בידית הצינור, אינן חוזרות עד שמערכת ההפעלה מסיימת את הפעולה. התוכנית יכולה להשתמש רק בפונקציות ReadFileEx ו- WriteFileEx עם ידית הצינור במצב חפיפה. הפונקציות ReadFile, WriteFile, CreateNamedPipe, ו- TransactNamedPipe יכולות לפעול בצורה סינכרונית או כפעולות חפיפה. פרמטר זה יכול להכיל צירוף כלשהו של דגלי מצב אבטחת גישה שמפורטים בטבלה זו. דגלי מצב אלה יכולים להיות שונים עבור מופעים שונים של אותו צינור. באפשרותך להגדיר אותם מבלי לדאוג למצבים אחרים שהגדרת כבר ב- dwOpenMode.
WRITE_DAC	לתהליך הקורא יש גישת כתיבה לרשימת בקרת הגישה, ACL (Access Control List) ששולטת בצינור בעל השם.
WRITE_OWNER	לתהליך הקורא יש גישת כתיבה לאב של הצינור בעל השם.
ACCESS_SYSTEM_SECURITY	התהליך הקורא הולך לקבל גישת כתיבה למערכת רשימת בקרת הגישה (ACL) של הצינור.

הפרמטר dwPipeMode מאפשר להגדיר כיצד להשתמש במופע הנוכחי של הצינור. באפשרותך להגדיר סוג, מצב קריאה, ומצב המתנה בפרמטר dwPipeMode. טבלה 17.21 מפרטת את ערכי המצב האפשריים.

טבלה 17.21: הערכים האפשריים עבור הפרמטר **dwPipeMode**.

מצב	תיאור
PIPE_TYPE_BYTE	**כתיבת** נתונים לצינור כ**זרם של בתים**. אינך יכול להשתמש במצב זה עם PIPE_READMODE_MESSAGE.
PIPE_TYPE_MESSAGE	**כתיבת** נתונים לצינור כ**זרם של הודעות**. אפשר להשתמש במצב זה עם PIPE_READMODE_MESSAGE או PIPE_READMODE_BYTE.

מצב	תיאור
PIPE_READMODE_BYTE	**קריאה** נתונים מהצינור כ**זרם של בתים**. אפשר להשתמש במצב זה עם PIPE_TYPE_BYTE או PIPE_TYPE_MESSAGE. אפשר להגדיר מצבי קריאה שונים עבור מופעים שונים של אותו צינור. אם מגדירים אפס, הפרמטר מקבל את ערך ברירת המחדל שהיא מצב קריאת בתים (Byte-Read Mode).
PIPE_READMODE_MESSAGE	**קריאה** נתונים מהצינור כ**זרם של הודעות**. תוכל להשתמש במצב זה רק אם תגדיר את PIPE_TYPE_MESSAGE. אפשר להגדיר מצבי קריאה שונים עבור מופעים שונים של אותו צינור. אם מגדירים אפס, הפרמטר מקבל את ערך ברירת המחדל, שהינה **מצב קריאת בתים** (Byte-Read Mode).
PIPE_WAIT	מאפשר **מצב נעילה** (Blocking Mode). כאשר מגדירים את ידית הצינור בפונקציות ReadFile, WriteFile, או ConnectNamedPipe – מערכת ההפעלה אינה מסיימת את הפעולות עד שהפונקציה קוראת נתונים, כותבת את כל הנתונים, או מחברת לקוח, בהתאמה. משמעות מצב PIPE_WAIT יכולה להיות המתנה עד אינסוף במצבים מסוימים, עד שתהליך לקוח יסיים לבצע את משימתו. אפשר להגדיר מצבי המתנה שונים עבור מופעים שונים של אותו צינור. אם מגדירים אפס, הפרמטר מקבל ערך ברירת המחדל שהינו מצב חסימה.
PIPE_NOWAIT	מאפשר **מצב אי-נעילה** (Non-Blocking Mode). במצב זה, הפונקציות ReadFile, WriteFile ו-ConnectNamedPipe תמיד חוזרות מיד.
PIPE_ACCESS_DUPLEX	הצינור הוא **דו-כיווני** (Bi-Directional) ; תהליכי השרת והלקוח יכולים לקרוא מהצינור ולכתוב בו. מצב זה מקנה לשרת גישה לצינור ששקולה ל- GENERIC_READ | GENERIC_WRITE. הלקוח יכול להגדיר GENERIC_READ או GENERIC_WRITE, או שניהם, כאשר הוא מתחבר לצינור על ידי השימוש בפונקציה CreateFile. באפשרותך להגדיר מצבי גישה שונים עבור מופעים שונים של אותו צינור.

מצב	תיאור
PIPE_ACCESS_INBOUND	זרימת הנתונים בצינור היא **מלקוח לשרת** בלבד. מצב זה מקנה לשרת גישה לצינור ששקולה ל- GENERIC_READ. הלקוח חייב להגדיר גישה GENERIC_WRITE כאשר הוא מתחבר לצינור.
PIPE_ACCESS_OUTBOUND	זרימת הנתונים בצינור עוברת **משרת ללקוח** בלבד. מצב זה מקנה לשרת גישה לצינור ששקולה ל- GENERIC_WRITE. הלקוח חייב להגדיר גישה GENERIC_READ כאשר מתחבר לצינור.

אם הפונקציה CreateNamedPipe מצליחה, היא מחזירה ידית לקצה השרת של הצינור בעל השם; ואם היא נכשלת, היא מחזירה INVALID_HANDLE_VALUE. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError. הפונקציה מחזירה ERROR_INVALID_PARAMETER אם הפרמטר nMaxInstances גדול מ- PIPE_UNLIMITED_INSTANCES.

כדי להשתמש ב- CreateNamedPipe ליצירת מופע של צינור בעל שם, חייבת להיות למשתמש גישה FILE_CREATE_PIPE_INSTANCE לאובייקט הצינור בעל השם. אם הפונקציה CreateNamedPipe יוצרת צינור חדש בעל שם, **רשימת בקרת השגיאה** (ACL) מפרמטר תכונות האבטחה מגדירה את השליטה בבקרת הגישה של הצינור בעל השם.

כל המופעים של צינור בעל שם חייבים להגדיר את אותו סוג צינור (Byte-Type, Message-Type), גישה לצינור (Inbound ,Duplex, או Outbound), מונה מופעים וערך פסק זמן (Time-Out). אם המופעים משתמשים בערכים שונים, CreateNamedPipe נכשלת, ו- GetLastError מחזירה ERROR_ACCESS_DENIED.

גודל חוצצי הקלט והפלט הוא בגדר המלצה. במילים אחרות, גודל החוצץ הממשי אשר Windows שומרת לכל אחד מקצות הצינור בעל השם הוא ברירת המחדל של המערכת, הערך המינימלי או המקסימלי של המערכת, או הגודל שאתה מגדיר מעוגל כלפי מעלה לגבול ההקצאה הבאה. שרת הצינור אינו צריך לחסום קריאה (Blocking Read) עד שהלקוח של הצינור מתחיל; אחרת, מצב תחרות יכול להתרחש. תנאי תחרות קורה בדרך כלל, כאשר קוד אתחול (כמו פונקציית אתחול של תהליך) חייב לנעול ולבדוק ידיות שעברו בירושה. **תנאי תחרות** (Race Condition) הוא שגיאה בתהליך מרובה מטלות, שבו קוד של מטלה אחת נשען על מטלה אחרת שתסיים פעולה כלשהי, ואין סינכרון בין שתי המטלות. התהליך פועל כראוי, כאשר המטלה השנייה "מנצחת" בתחרות על ידי סיום פעולתה לפני שהמטלה הראשונה צריכה אותה; אבל התהליך נכשל אם המטלה הראשונה "מנצחת" בתחרות.

התוכנית תמיד מוחקת מופע של צינור בעל שם, כאשר היא סוגרת את הידית האחרונה של אותו מופע של הצינור. בסעיף 17.34 תלמד להשתמש ב- CallNamedPipe ו- ConnectNamedPipe יחד עם CreateNamedPipe כדי לבצע פלט בשרת רשת.

17.33 התחברות לצינור בעל שם

למדת בסעיף קודם שבאפשרות התוכניות להשתמש בפונקציה CreateNamedPipe כדי ליצור צינור בעל שם, או להתחבר אליו. פעמים רבות, התוכנית שלך תפעל במחשב הלקוח, אשר מכריח אותה להתחבר לצינור בעל שם כדי להתקשר עם השרת, במקום ליצור צינור בעל שם (פעולה החייבת להתבצע בשרת הצינור). בנוסף לכך, לפני שהתוכנית שלך יכולה להתחבר לצינור בעל השם של השרת, הצינור בעל השם חייב לקרוא לפונקציה ConnectNamedPipe. הפונקציה ConnectNamedPipe מורה לתהליך שרת צינור בעל שם להמתין לתהליך לקוח, כדי שיתחבר למופע של הצינור בעל שם. תהליך הלקוח קורא לפונקציה CreateFile או לפונקציה CallNamedPipe כדי להתחבר למופע. תלמד יותר על הפונקציה CallNamedPipe בסעיף הבא. את הפונקציה ConnectNamedPipe כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL ConnectNamedPipe(
  HANDLE hNamedPipe,          // handle to named pipe to connect
  LPOVERLAPPED lpOverlapped // pointer to overlapped structure
);
```

הפרמטר hNamedPiipe מציין את צד השרת במופע צינור בעל שם, והפונקציה CreateNamedPipe מחזירה ידית זו. הפרמטר lpOverlapped מצביע למבנה מסוג OVERLAPPED (וכפי שלמדת בסעיף 17.2, הוא מאפשר לתוכניות לבצע קלט/פלט אסינכרוני). אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, היא מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError.

באפשרות תהליך שרת של צינור בעל שם להשתמש ב-ConnectNamedPipe עם מופע חדש שנוצר לצינור, או עם מופע שהתחבר קודם לכן עם תהליך לקוח אחר. במקרה האחרון, תהליך השרת חייב לקרוא תחילה לפונקציה DisconnectNamedPipe כדי לנתק את חיבור הידית עם הלקוח הקודם, לפני ש-ConnectNamedPipe יכולה לחבר את הידית הזו אל לקוח חדש. אחרת, ConnectNamedPipe מחזירה False, ו-GetLastError מחזירה ERROR_NO_DATA אם הלקוח הקודם סגר את הידית שלו, או שהיא מחזירה ERROR_PIPE_CONNECTED אם לא סגר את הידית שלו.

התנהגות הפונקציה ConnectNamedPipe תלויה בשני תנאים: אם אתה קובע את מצב ההמתנה של ידית הצינור ל**נעילה** (Blocking) או **אי-נעילה** (Non-Blocking), ואם אתה מורה לפונקציה שתפעל בצורת מצב סינכרונית או מצב חפיפה. השרת מגדיר בשלב האתחול את מצב ההמתנה לידית הצינור בפונקציה CreateNamedPipe, וצריך להשתמש בפונקציה SetNamedPipeHandleState כדי לשנות את המצב.
אם הפונקציה פותחת את hNamedPipe עם FILE_FLAG_OVERLAPPED, אסור שהפרמטר lpOverlapped יהיה שווה ל-NULL; הוא חייב להצביע למבנה OVERLAPPED

תקף. אם הפונקציה פותחת hNamedPipe עם FILE_FLAG_OVERLAPPED, ו-lpOverlapped שווה ל-NULL, הפונקציה ConnectNamedPipe יכולה לדווח בצורה לא נכונה שפעולת ההתחברות הסתיימה. מצד שני, אם הפונקציה יוצרת hNamedPipe עם FILE_FLAG_OVERLAPPED ו-lpOverlapped אינו שווה ל-NULL, המבנה OVERLAPPED שמוצבע על ידי הפרמטר lpOverlapped חייב להכיל ידית לאובייקט אירוע שמאופס ידנית (Manual-Reset Event Object), אשר השרת יכול להשתמש בפונקציה CreateEvent כדי ליצור אותו).

אם הפונקציה אינה פותחת hNamedPipe עם FILE_FLAG_OVERLAPPED, ו-lpOverlapped שווה ל-NULL, הפונקציה אינה חוזרת עד שהתוכנית מתחברת ללקוח, או שנוצר מצב שגיאה. פעולות סינכרוניות מוצלחות גורמות לפונקציה להחזיר True אם לקוח מתחבר לאחר שהתוכנית קוראת לפונקציה. אם לקוח מתחבר לפני שהתוכנית קוראת לפונקציה, הפונקציה מחזירה False ו-GetLastError מחזירה ERROR_PIPE_CONNECTED. תוצאת False יכולה לקרות, אם לקוח מתחבר במרווח זמן שבין הקריאה ל-CreateNamedPipe לבין הקריאה ל-ConnectNamedPipe. במקרה זה, יש חיבור טוב בין הלקוח והשרת, אפילו אם הפונקציה מחזירה False.

אם הפונקציה אינה פותחת את hNamedPipe עם FILE_FLAG_OVERLAPPED, ו-lpOverlapped אינו שווה ל-NULL, הפעולה מתבצעת בצורה אסינכרונית. הפונקציה חוזרת מיד עם ערך ששווה ל-False. אם תהליך לקוח מתחבר לפני שהתוכנית קוראת לפונקציה, GetLastError מחזירה ERROR_PIPE_CONNECTED ; אחרת, GetLastError מחזירה ERROR_IO_BENDING, אשר מציין שהפעולה מתבצעת ברקע. כאשר דבר זה מתרחש, התוכנית קובעת את אובייקט האירוע שבמבנה OVERLAPPED למצב לא מסומן (Non-Signaled State), לפני ש-ConnectNamedPipe חוזרת, והתוכנית קובעת אותו למצב מסומן כשהלקוח מתחבר למופע זה של הצינור.

באפשרות תהליך השרת להשתמש בכל פונקציה מפונקציות ההמתנה או בפונקציה SleepEx, כדי לקבוע מתי מצב אובייקט האירוע הוא מסומן (Signaled). אחר כך, השרת יכול להשתמש בפונקציה GetOverlappedResult כדי לקבוע את תוצאת ConnectNamedPipe. אם ידית הצינור שאתה מגדיר היא במצב אי-נעילה (Non-Blocking Mode), ConnectNamedPipe תמיד חוזרת מיד. במצב אי-נעילה, ConnectNamedPipe מחזירה True בפעם הראשונה שהתוכנית קוראת לה עבור מופע צינור, אשר התהליך כבר ניתק אותו מלקוח קודם. הדבר מציין למערכת שהצינור תקף עכשיו, כדי לחבר אותו עם תהליך לקוח חדש. בכל שאר המקרים, כאשר ידית הצינור במצב אי-נעילה, ConnectNamedPipe מחזירה False. במקרים אלה, GetLastError מחזירה ERROR_PIPE_LISTENING אם אין לקוח שמחובר, ERROR_PIPE_CONNECTED - אם יש לקוח מחובר, ו- ERROR_NO_DATA - אם לקוח קודם סגר את ידית הצינור שלו אבל השרת לא התנתק. שים לב, כאשר צינור נמצא במצב אי-נעילה, חיבור טוב בין לקוח לבין שרת מתקיים רק לאחר שהתוכנית מקבלת את הודעת השגיאה ERROR_PIPE_CONNECTED.

✍ **הערה:** הפונקציה ConnectNamedPipe תומכת במצב אי-נעילה (Non-Blocking Mode) בגלל תאימות עם Microsoft LAN Manager 2.0. אינך חייב להשתמש בה כדי להשיג קלט ופלט אסינכרוני עם צינורות בעלי שם.

17.34 קריאה לצינור בעל שם

תוכניות יכולות ליצור צינור בעל שם בקצה השרת שבחיבור רשת. קצה הלקוח חייב במקרה זה להתחבר לצינור בעל השם כדי לשלוח אליו מידע. באפשרותך להשתמש בתוכניות בפונקציה CreateFile או ב-CallNamedPipe כדי להתחבר לצינור בעל שם. הפונקציה CallNamedPipe מבצעת חיבור לצינור מ**סוג הודעה** (Message-Type) וממתינה אם מופע הצינור אינו זמין, כותבת בצינור וקוראת ממנו, ואחר כך סוגרת את הצינור. את הפונקציה CallNamedPipe כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL CallNamedPipe(
  LPCTSTR lpNamedPipeName,   // pointer to pipe name
  LPVOID lpInBuffer,         // pointer to write buffer
  DWORD nInBufferSize,       // size, in bytes, of write buffer
  LPVOID lpOutBuffer,        // pointer to read buffer
  DWORD nOutBufferSize,      // size, in bytes, of read buffer
  LPDWORD lpBytesRead,       // pointer to number of bytes read
  DWORD nTimeOut             // time-out time, in milliseconds
);
```

הפונקציה CallNamedPipe מקבלת את הפרמטרים שמפורטים בטבלה 17.22.

טבלה 17.22: הפרמטרים שמקבלת הפונקציה **CallNamedPipe**.

פרמטר	תיאור
lpNamedPipeName	מצביע למחרוזת המסתיימת ב-NULL ומגדירה את שם הצינור.
lpInBuffer	מצביע לחוצץ שמכיל את הנתונים אשר CallNamedPipe כותבת בצינור.
nInBufferSize	הגודל, בבתים, של חוצץ הכתיבה.
lpOutBuffer	מצביע לחוצץ שמקבל את הנתונים ש- CallNamedPipe קוראת מהצינור.
nOutBufferSize	הגודל, בבתים, של חוצץ הקריאה.

פרמטר	תיאור
lpBytesRead	מצביע למשתנה בן 32 סיביות שמקבל את מספר הבתים ש- CallNamedPipe קוראת מהצינור.
nTimeOut	מספר המילישניות שצריך להמתין לצינור בעל השם עד שיהיה זמין. בנוסף לערכים מספריים, התוכנית יכולה לציין ערכים מיוחדים שמפורטים בטבלה 17.23.

כאשר אתה קורא לצינור בעל שם דרך רשת, חשוב לקבוע מגבלה לפרק הזמן שבו התוכניות ימתינו לצינור בעל השם עד שיגיב. הגבלות הזמן מונעות ממחשב לקוח להמתין עד אינסוף, כדי להתחבר עם צינור בעל שם שיכול להיות עסוק, מנותק, או אפילו שאינו קיים. בנוסף להגדרת זמן המתנה קבוע במילישניות של התוכניות שלך לתגובה מצינור בעל שם, תוכל להשתמש גם בערכים שמפורטים בטבלה 17.23, כדי לשלוט בזמן שתהליך הלקוח שלך צריך להמתין כדי להתחבר עם צינור בעל שם.

טבלה 17.23: קבועים של פרקי זמן להמתנה עבור הפונקציה **CallNamedPipe**.

פרמטר	תיאור
NMPWAIT_NOWAIT	אינו ממתין לצינור בעל שם. אם הצינור בעל שם אינו זמין, הפונקציה מחזירה שגיאה.
NMPWAIT_WAIT_FOREVER	ממתין לצמיתות.
NMPWAIT_USE_DEFAULT_WAIT	משתמש בפרק הזמן של ברירת המחדל שמוגדר בקריאה לפונקציה CreateNamedPipe.

אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, היא מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError.

הקריאה ל-CallNamedPipe שקולה לקריאה לפונקציות CreateFile (או WaitNamedPipe, אם CreateFile אינה יכולה לפתוח את הצינור מיד), ל-TransactNamedPipe ול-CloseHandle. התוכנית קוראת ל-CreateFile עם דגל גישה GENERIC_READ | GENERIC_WRITE, דגל ידית שעברה בירושה וערכו False, ומצב שיתוף (Share Mode) שערכו אפס (שמציין שאין שיתוף במופע הצינור הזה). אם ההודעה שתהליך השרת כותב בצינור ארוכה מ-nOutBufferSize, CallNamedPipe מחזירה False, GetLastError מחזירה ERROR_MORE_DATA. תהליך השרת מתעלם משארית ההודעה, מכיון ש-CallNamedPipe סוגרת את הידית של הצינור לפני שהיא חוזרת.

בתקליטור שמצורף לספר זה תמצא את התוכניות **Server.c** ו-**Client.c** (בתיקיה **np_client_server**\Books\59285\Chap17). כאשר מהדרים ומריצים את התוכנית **Server.c**, היא יוצרת צינור בעל שם במחשב המקומי. אם אחר כך מהדרים ומריצים את התוכנית **Client.c**, היא **Client.c** קוראת לצינור בעל שם. התוכנית **Client.c**

מציגה את ההודעה ואחר כך כותבת מחרוזת נתונים בצינור בעל שם. כאשר השרת מקבל את ההודעה מהצינור בעל שם, הוא מציג את המידע בתהליך משלו. שים לב שדוגמה זו פועלת על מחשב בודד שמייצג תחנת שרת ותחנת לקוח גם יחד.

✍ **הערה:** הפונקציה CallNamedPipe נכשלת, אם הצינור שהפונקציה מנסה לקרוא לו הוא צינור מסוג-בית (Byte-Type Pipe).

17.35 התנתקות מצינור בעל שם

תוכניות משתמשות בצינורות בעלי שם, כדי לקשר בין שני תהליכים שפועלים בשני מחשבים שונים (ואפילו באותו מחשב). לאחר שמסיימים את שיתוף הנתונים בין שני התהליכים, חשוב לסגור את הצינור בעל שם, כך שלא יגרום להאטת הרשת או המחשב. בדרך כלל, סוגרים את ידית הצינור בעל שם אצל הלקוח תחילה, ואחר כך מנתקים את הצינור בעל שם אצל השרת. הפונקציה DisconnectNamedPipe מנתקת את קצה השרת של מופע צינור בעל שם מתהליך לקוח. את הפונקציה DisconnectNamedPipe כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL DisconnectNamedPipe(HANDLE hNamedPipe);
```

הפרמטר hNamedPipe מציין מופע של צינור בעל שם, והפונקציה CreateNamedPipe חייבת ליצור ידית זו. אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; ואם הפונקציה נכשלת, היא מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError.

אם קצה הלקוח של צינור בעל שם פתוח, הפונקציה DisconnectNamedPipe כופה סגירה על הקצה הזה של הצינור בעל שם. הלקוח מקבל הודעת שגיאה אם ינסה לפנות שוב אל הצינור. לקוח אשר DisconnectNamedPipe מנתקת אותו מהצינור חייב עדיין להשתמש בפונקציה CloseHandle כדי לסגור את קצה הצינור שלו.

כאשר תהליך השרת מנתק מופע צינור, הוא מתעלם מכל הנתונים שלא נקראו מהצינור. לפני ההתנתקות, ובכדי להיות בטוח שנתונים לא אובדים, השרת צריך לקרוא לפונקציה FlushFileBuffers. פונקציה זו אינה חוזרת, עד שתהליך הלקוח קורא את כל הנתונים. תהליך השרת חייב לקרוא ל-DisconnectNamedPipe כדי לנתק את ידית הצינור מהלקוח הקודם שלה, לפני שהוא יכול לחבר את הידית ללקוח אחר על ידי השימוש בפונקציה ConnectNamedPipe.

17.36 בחינה חוזרת של העיבוד האסינכרוני

למדת שבאפשרות התוכניות שלך לאחסן נתונים בקבצים ולאחזר מהם נתונים במרבית ההתקנים שמשתמשים בפלט סינכרוני או אסינכרוני. כמו שלמדת, קלט/פלט אסינכרוני של קבצים מאפשר לתוכניות לבצע פעולות שאינן סינכרוניות (Non-Synchronized). עם זאת, חשוב להבין יותר טוב מדוע תרצה לבצע פעולות קלט/פלט אסינכרוניות.

בהשוואה למרבית הפעולות שמבצע המחשב, קלט/פלט מהתקנים ואליהם, בהתאמה, הוא אחד האיטיים ביותר. המעבד מבצע פעולות אריתמטיות ואפילו צביעת המסך יותר מהר מאשר הוא קורא נתונים מקובץ או כותב נתונים בקובץ ישירות, או דרך רשת. קלט/פלט אסינכרוני מאפשר לך להשתמש במטלות רבות, כדי להורות למערכת ההפעלה שתקרא מהתקן או תכתוב בהתקן, בזמן ששאר הקוד ביישום ממשיך להתבצע.

כדי להבין יותר טוב כיצד קלט/פלט אסינכרוני משפר את ביצועי התוכנית, נניח שאתה מפתח יישום מסד נתונים פשוט. כאשר המשתמש פותח מסד נתונים, היישום קורא את תכולת מסד הנתונים לזיכרון, וגם אל קובץ אינדקס כלשהו. לאחר שהמשתמש בוחר במסד הנתונים שעליו לטעון, היישום חייב להיות מופסק זמנית כדי לטעון את כל הנתונים לתוך הזיכרון (לפעמים הוא מציג שעון חול לנוחות המשתמש).

כשאתה משתמש בקלט/פלט אסינכרוני, התוכנית יכולה להתחיל בפעולת הקלט/פלט של הדיסק שמבוצעת על ידי בקר הדיסק, והמעבד מבצע משימות אחרות שאינן קשורות באותו זמן. קלט/פלט אסינכרוני מאפשר לך לבצע פעילויות קלט/פלט רבות באותו זמן - או להתחיל בפעילות קלט/פלט שאינה קריטית (Non-Critical I/O) ולאפשר לה להסתיים בזמן שלה מבלי להאט את ביצועי התוכנית שלך. ככל שהתוכניות שלך הופכות להיות יותר מורכבות ואתה קורא יותר נתונים מהכונן הקשיח וכותב בו יותר נתונים, או שאתה פועל עם התקנים אחרים, יתרונות הקלט/פלט האסינכרוני הופכים למשמעותיים מאוד בשבילך.

17.37 קלט פלט אסינכרוני

כדי לגשת להתקן בצורה אסינכרונית, אתה חייב לקרוא תחילה ל- CreateFile כדי לפתוח את ההתקן ולהגדיר את הדגל FILE_FLAG_OVERLAPPED בפרמטר dwFlagsAndAttrs. הדגל FILE_FLAG_OVERLAPPED מודיע למערכת שבכוונתך להשתמש בהתקן עבור קלט/פלט אסינכרוני.

מערכת ההפעלה Win32 מאפשרת לך להשתמש בארבע טכניקות שונות, כדי לבצע קלט/פלט אסינכרוני. התיאוריה ליישום הטכניקות האלו היא אחת. כאשר אתה מבצע קלט/פלט אסינכרוני, אתה חייב לקבל דרישת קלט/פלט ממערכת ההפעלה. מערכת ההפעלה מכניסה כל הדרישות לקלט/פלט לתור, ומטפלת בהן בצורה פנימית. בזמן שמערכת ההפעלה מטפלת בדרישות קלט/פלט, היא מאפשרת למטלה שלך לחזור ולהמשיך בעיבוד שלה. בנקודה כלשהי אחר כך מערכת ההפעלה מסיימת את משימת הקלט/פלט ומודיעה ליישום שלך ששלחה וקיבלה את הנתונים, או שקרתה שגיאה.

כמו שנאמר, יש ארבע טכניקות קלט/פלט שונות, שמפורטות בטבלה 17.24 לפי סדר המורכבות שלהן, מהקל ביותר להבנה וליישום (**סימון ידית התקן**, Device Handle Signaling) עד הקשה ביותר להבנה וליישום (**ערוצי קלט/פלט משלימים**, I/O Completion Ports).

טבלה 17.24: ארבע הטכניקות לטיפול בקלט/פלט אסינכרוני.

טכניקה	תיאור
Signaling a device object (**סימון אובייקט התקן**)	שימוש ב**פונקציית המתנה** (Wait Function) וידית התקן כדי לבצע קלט/פלט אסינכרוני. טכניקה זו אינה שימושית אם אתה מתכוון לבצע דרישות קלט/פלט רבות בו-זמנית בהתקן יחיד. מטלה אחת יכולה לעסוק בדרישות קלט/פלט, ומטלה אחרת - לטפל בו.
Signaling an event object (**סימון אובייקט אירוע**)	שימוש בפונקציה WaitForMultipleObjects וב**אובייקט אירוע** (Event Object) אחד או יותר, כדי לבצע קלט/פלט אסינכרוני. מטלה אחת עשויה להוציא דרישה לקלט/פלט, ומטלה אחרת - לטפל בו.
Alertable I/O (**הודעות על פעולות קלט/פלט**)	שימוש בתור הודעות מיוחד, כדי לטפל בהודעות שינוי מצב של מערכת ההפעלה על כך שפעולות קלט/פלט אסינכרוניות הושלמו. כאן יש יותר גמישות מאשר בעת שימוש באובייקט אירוע גרעין, מכיון שבאפשרותך להשתמש ב**פונקציות משוב** (Callback Functions) ולטפל באופן ייחודי בהודעות, כדי להגיב למידע מערכת ההפעלה על פעולת הקלט/פלט. המטלה שהציגה את דרישת הקלט/פלט חייבת גם כן לטפל בתגובה. בטכניקה זו אפשר להשתמש במערכות Windows NT בלבד.
I/O completion ports (**ערוצי קלט/פלט משלימים**)	השימוש ב-concurrent threading model (**מודל טיפול בו-זמנית על ידי מטלות**, מוסבר בסעיף 17.46) מכוון להגיב למספר גדול של דרישות קלט/פלט בו- זמנית. הדבר מאפשר למטלה אחת לעסוק בדרישת קלט/פלט ומטלה אחרת - לטפל בו. הטכניקה המדורגת של ערוצים משלימים, היא השימושית ביותר בין מפתחים מקצועיים. באפשרותך להשתמש בטכניקה זו במערכות Windows NT בלבד.

17.38 המבנה OVERLAPPED

בטבלה 17.24 למדת שהטכניקה הפשוטה ביותר לביצוע קלט/פלט אסינכרוני בהתקן היא להשתמש ב**סימון ידית התקן** (Device-Handle Signaling), אשר נדון בה בסעיף 17.39. כדי להציג דרישת קלט/פלט, צריך להשתמש בפונקציות ReadFile ו-WriteFile שהוצגו בסעיפים 17.7 ו-17.8. עם זאת, כדי לבצע קלט/פלט אסינכרוני בהתקן, התוכניות חייבות לאתחל מבנה מסוג OVERLAPPED ולמסור את הכתובת שלו בפרמטר lpOverlapped. ממשק Win32 API מגדיר את המבנה OVERLAPPED, כמו שרואים להלן:

```
typedef struct _OVERLAPPED {
  DWORD Internal;
  DWORD InternalHigh;
  DWORD Offset;
  DWORD OffsetHigh;
  HANDLE hEvent;
} OVERLAPPED;
```

טבלה 17.25 מפרטת את איברי המבנה OVERLAPPED.

טבלה 17.25: איברי המבנה OVERLAPPED.

איבר	תיאור
Internal	מגדיר מצב **תלוי מערכת** (System-Dependent). איבר זה תקף כאשר הפונקציה GetOverlappedResult חוזרת מבלי לקבוע את מידע השגיאה המורחב ל- ERROR_IO_PENDING. שמור לשימוש מערכת ההפעלה.
InternalHigh	מגדיר את אורך הנתונים שמועברים. איבר זה תקף כאשר הפונקציה GetOverlappedResult מחזירה True. שמור לשימוש מערכת ההפעלה.
Offset	מגדיר מקום בקובץ שממנו צריך להתחיל בהעברה. מקום בקובץ הוא היסט (Offset) הנמדד בבתים מתחילת הקובץ. התהליך הקורא קובע איבר זה לפני הקריאה לפונקציות ReadFile או WriteFile. התהליך הקורא מתעלם מאיבר זה כאשר קוראים מצינור בעל שם ומהתקני תקשורת, או כותבים אליהם.
OffsetHigh	מגדיר את המילה הגבוהה (High Word) של ההיסט שממנה צריך להתחיל בהעברה. התהליך הקורא מתעלם מאיבר זה כאשר קוראים מצינור בעל שם ומהתקני תקשורת, או כותבים אליהם.
hEvent	מזהה אירוע שנקבע ל**מצב מסומן** (Event Set To Signaled State) כאשר ההעברה מסתיימת. התהליך הקורא קובע איבר זה לפני שהוא קורא לאחת הפונקציות ReadFile, WriteFile, ConnectNamedPipe, או TransactNamedPipe.

באפשרותך להשתמש במאקרו HasOverlappedIoCompleted כדי לקבוע אם פעולת קלט/פלט אסינכרוני הסתיימה. בפונקציה CancelIo תוכל להשתמש כדי לבטל פעולת קלט/פלט אסינכרוני.

17.39 קלט/פלט אסינכרוני עם אובייקט התקן גרעין

התוכניות יכולות לבצע קלט/פלט אסינכרוני על ידי שימוש בארבע טכניקות שונות, שביניהן, הקלה ביותר לשימוש היא **אובייקט התקן גרעין** (Device Kernel Object). כאשר מבצעים קלט/פלט אסינכרוני עם אובייקט התקן גרעין, אתה למעשה מורה למטלה להמתין עד שהקלט/פלט יסתיים. לדוגמה, נניח שאתה קורא מקובץ על ידי שימוש בפונקציה ReadFile ובמהלך הקריאה אתה מבצע איזשהו עיבוד. עם זאת, התוכנית אינה יכולה להמשיך מעבר לנקודה מסוימת עד שפעולת הקריאה מסתיימת. במקרה כזה, אפשר לכתוב קוד שדומה לקטע שלהלן:

```
ReadFile(hFile, bBuffer, sizeof(bBuffer),
         &dwNumBytesRead, &Overlapped);
// processing here
WaitForSingleObject(hFile, INFINITE);
                         // Wait until all data is in the buffer
```

כאשר קוראים לפונקציה WaitForSingleObject עם ידית של התקן קלט/פלט אסינכרוני, הפונקציה ממתינה עד שמערכת ההפעלה מסיימת את העיבוד המתאים עבור ההתקן, לפני שהיא משחררת את המטלה ומאפשרת לה להמשיך.

17.40 הגדרת גודל שטחי עבודה (Working-Set Size Quotas)

כאשר התוכניות מבצעות קלט/פלט אסינכרוני, מערכת ההפעלה מחזיקה עבורן רשימת דרישות קלט/פלט לביצוע. מערכת ההפעלה מתקנת את גודל הרשימה בעת אתחול המערכת. לפעמים, דרישת קלט/פלט אסינכרוני יכולה להיכשל מפני שרשימת דרישות קלט/פלט לביצוע מלאה כבר. אם הרשימה מלאה כאשר אתה רוצה לצרף דרישה נוספת, ReadFile ו-WriteFile מחזירות False, ו-GetLastError מחזירה ERROR_INVALID_USER_BUFFER או ERROR_NOT_ENOUGH_MEMORY. יותר מכך, כאשר אתה מבקש דרישת קלט/פלט, המערכת חייבת "**לנעול דף**" ("Page Lock") - המתייחס לחוצץ הנתונים של התוכנית. חוצץ הנתונים הוא חלק **מקבוצת התצורה** (Working Set) של התהליך, ולכל תהליך יש קבוצת תצורה מקסימלית. **קבוצת התצורה** של תהליך היא קבוצת דפי הזיכרון שזמינים כרגע לתהליך ב-RAM הפיסי. אם אין לך די מקום בקבוצת התצורה של התהליך, הדרישה לקלט/פלט גורמת לכישלון,

ו-GetLastError מחזירה ERROR_NOT_ENOUGH_QUOTA. באפשרותך להגדיל את קבוצת התצורה של התהליך על ידי קריאה ל-SetProcessWorkingSetSize, שסעיף 17.41 מסביר אותה בפירוט.

17.41 שינוי גודל של שטחי עבודה

בסעיף קודם למדת שבאפשרות התוכניות להגדיל את קבוצת התצורה שלהן, ולמעשה - שטחי העבודה (קבוצת דפי הזיכרון שזמינים כרגע לתהליך הזיכרון), לפי צרכי התוכנית שדורשת מרחב נוסף בקבוצת התצורה שלה. דפים אלה שוכני זיכרון ותקפים לשימוש עבור היישום מבלי לגרום ולעורר שגיאת דף. גודל קבוצת התצורה של התהליך מוגדר בבתים. הגודל המינימלי והגודל המקסימלי של קבוצת התצורה משפיע על התנהגות דפי הזיכרון הווירטואלי של התהליך.

הפונקציה SetProcessWorkingSetSize קובעת את הגודל המינימלי ואת הגודל המקסימלי של קבוצת התצורה עבור התהליך שאתה מגדיר. את הפונקציה SetProcessWorkingSetSize כותבים בתוכניות כמו בהגדרה שלהלן:

```
BOOL SetProcessWorkingSetSize(
 HANDLE hProcess,  // open handle to the process of interest
 DWORD dwMinimumWorkingSetSize,
                   // specifies minimum working set size
 DWORD dwMaximumWorkingSetSize
                   // specifies maximum working set size
);
```

הפרמטר hProcess הוא ידית פתוחה לתהליך שרוצים לקבוע את קבוצת התצורה שלו. תחת Windows NT, הידית חייבת להיות בעלת זכות גישה מסוג PROCESS_SET_QUOTA. הפרמטר dwMinimumWorkingSetSize מגדיר קבוצת תצורה מינימלית עבור התהליך. מנהל הזיכרון הווירטואלי מנסה לשמור לפחות כמות זו של זיכרון במסגרת התהליך כל עוד שהתהליך פעיל. הפרמטר dwMaximumWorkingSetSize מגדיר קבוצת תצורה מקסימלית עבור התהליך. מנהל הזיכרון הווירטואלי מנסה לא לשמור יותר מכמות זו של זיכרון בתהליך, כל עוד התהליך פעיל ויש מחסור בזיכרון.

אם הפרמטרים dwMinimumWorkingSetSize ו- dwMaximumWorkingSetSize שווים ל-0xFFFFFFFF, הפונקציה מסדרת באופן זמני את הקצאת שטחי העבודה לאפס. הדבר גורם להעתקת כל הזיכרון של התהליך להתקן חיצוני, כאשר יש מחסור בזיכרון פיסי. אם הפונקציה מצליחה, הערך המוחזר שונה מאפס; אם הפונקציה נכשלת, היא מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError.

באפשרותך לרוקן את שטח העבודה של התהליך המוגדר על ידי הגדרת הערך 0xFFFFFFFF עבור שני הפרמטרים המינימלי והמקסימלי של גודל שטח העבודה הזה. אם אחד מהפרמטרים dwMinimumWorkingSetSize או dwMaximumWorkingSetSize גדול יותר מגודל שטח העבודה הנוכחי של התהליך, התהליך המוגדר חייב להיות בעל

הרשאה SE_INC_BASE_PRIORITY_NAME. מערכת ההפעלה מקצה שטחי עבודה בשיטת "ראשון נכנס, ראשון מקבל שירות" (First-Come, First-Served). לדוגמה, אם יישום מצליח לקבוע 40MB כגודל שטח העבודה המינימלי במערכת של 64MB, ויישום שני דורש 40MB כשטח עבודה, מערכת ההפעלה דוחה את דרישת היישום השני.

השימוש בפונקציה SetProcessWorkingSetSize כדי לקבוע את שטח העבודה המינימלי והמקסימלי אינו מבטיח שמערכת ההפעלה תשמור את כמות הזיכרון הנדרשת, או שהזיכרון יישאר זמין כל הזמן. כאשר היישום במצב המתנה (Idle), או כאשר יש מחסור בזיכרון (Low-Memory Situation) ויש דרישה לזיכרון נוסף, מערכת ההפעלה יכולה לצמצם את גודל שטח העבודה של היישום. היישום יכול להשתמש בפונקציה VirtualLock כדי לנעול תחומים במרחב הזיכרון הווירטואלי של היישום; אך הדבר עלול לגרום להורדת ביצועי המערכת באופן ניכר.

כאשר אתה מגדיל את שטח העבודה של היישום, אתה גוזל למעשה זיכרון פיסי משאר יישומי המערכת. הדבר יכול לגרום להורדת ביצועי יישומים אחרים ושל המערכת כולה. גזילת זיכרון פיסי עלולה לגרום גם לכישלון בפעולות שדורשות זיכרון פיסי להמשך פעולתן; לדוגמה, יצירת תהליכים, מטלות, ומאגר גרעין (Kernel Pool). לכן, כאשר אתה מתכנן יישום, אתה חייב להיות זהיר בעת שימוש בפונקציה SetProcessWorkingSetSize, ותמיד לקחת בחשבון את ביצועי כלל המערכת.

17.42 הפונקציה GetLastError

במהלך הלימוד ראית פעמים רבות שאפשר להשיג מידע רב יותר אודות שגיאות התוכנית. בדרך כלל, התוכניות צריכות לקרוא לפונקציה GetLastError כדי להשיג מידע נוסף. פונקציה זו מחזירה את ערך קוד השגיאה האחרונה של המטלה הקוראת. מערכת ההפעלה מחזיקה את ערך קוד השגיאה האחרונה על פי מטלות. על כן, במצב של ריבוי מטלות, מטלה אחת אינה **דורסת** (Overwrite) את קוד השגיאה האחרונה של מטלה אחרת. את הפונקציה GetLastError כותבים כמו בהגדרה שלהלן:

```
DWORD GetLastError(void);
```

הפונקציה מחזירה את קוד השגיאה האחרונה של המטלה הקוראת. הפונקציות קוראות לפונקציה SetLastError כדי לקבוע את ערך קוד השגיאה האחרונה. צריך לקרוא לפונקציה GetLastError מיד לאחר שהערך המוחזר של פונקציה מציין שקריאה ל-GetLastError מחזירה נתונים בעלי משמעות (כמו למשל, מידע שגיאה מורחב), מכיון שחלק מהפונקציות קוראות ל- SetLastError(0) כאשר הן מצליחות, ומוחקות את קוד השגיאה שנקבע על ידי הפונקציה האחרונה שנכשלה.

מרבית הפונקציות ממשק התכנות Win32 API אשר קובעות את ערך קוד השגיאה האחרונה של המטלה, עושות זאת כאשר הן נכשלות; חלק קטן מהפונקציות קובעות אותו כאשר הן מצליחות. ערך קוד השגיאה המוחזר, כמו False, NULL, 0xFFFFFFFF, או 1-, מציין בדרך כלל כישלון של הפונקציה. קודי שגיאה הם ערכים בני 32 סיביות

(סיבית 31 היא הסיבית המשמעותית ביותר). סיבית 29 שמורה עבור שגיאות שמוגדרים על ידי היישומים; אין קוד שגיאת מערכת שקובע את הסיבית הזו. אם אתה מגדיר קוד שגיאה עבור היישום, קבע את סיבית 29 לאחד. קביעה זו מציינת שהיישום מגדיר את קוד השגיאה, ומבטיח שקוד השגיאה שלך אינו מתנגש עם קודי השגיאה המוגדרים במערכת ההפעלה. באפשרותך להשתמש בפונקציה FormatMessage כדי לפרמט את פלט התוצאה שמתקבל מקריאה ל-GetLastError. בסעיף הבא תמצא הסבר של הפונקציה FormatMessage.

17.43 עריכת הודעות שגיאה עם FormatMessage

התוכניות משתמשות בפונקציה GetLastError כדי להציג את השגיאה האחרונה של המטלה. הן מציגות ערך מספרי בלבד, בשעה שאנו מעדיפים לראות כמעט תמיד את תיאור השגיאה כהודעת טקסט ברורה ומובנת.

הפונקציה FormatMessage עורכת מחרוזות הודעה. את הגדרת ההודעה היא יכולה לקבל מחוצץ שמועבר אליה, או מטבלת הודעות במודול שכבר נטען. בנוסף לכך, התהליך הקורא עשוי לבקש מהפונקציה לחפש בטבלאות של הודעות המערכת את הגדרת ההודעה המבוקשת. הפונקציה מוצאת את הגדרת ההודעה בטבלת הודעות על פי מזהה ההודעה ומזהה השפה. הפונקציה מעתיקה את טקסט ההודעה הערוכה אל חוצץ פלט ומבצעת מספר פקודות, אם יש כאלו בגוף ההודעה. את הפונקציה FormatMessage כותבים בתוכניות כמו בהגדרה שלהלן:

```
DWORD FormatMessage(
  DWORD dwFlags,            // and processing options
  LPCVOID lpSource,         // pointer to message source
  DWORD dwMessageId,        // requested message identifier
  DWORD dwLanguageId,       // language identifier for
                            // requested message
  LPTSTR lpBuffer,          // pointer to message buffer
  DWORD nSize,              // maximum size of message buffer
  va_list *Arguments        // address of array of message
);                          // inserts
```

טבלה 17.26 מפרטת את הפרמטרים שמקבלת הפונקציה FormatMessage.

טבלה 17.26: הפרמטרים שמקבלת הפונקציה **FormatMessage**.

פרמטר	תיאור
dwFlags	קבוצה של דגלי סיביות, אשר מגדירות את היבטי תהליך העריכה ואיך לפרש את הפרמטר lpSource. בית הסדר הנמוך (Low-Order Byte) של dwFlags מגדיר כיצד הפונקציה תטפל

פרמטר	תיאור
	במעברי שורות (Line Breaks) בחוצץ הפלט. בית הסדר הנמוך גם יכול להגדיר את הרוחב המקסימלי של פלט שורה מפורמט. אפשר להגדיר צירוף של דגלי סיביות כמפורט בטבלה 17.27.
lpSource	מקום ההגדרות של ההודעה. סוג פרמטר זה תלוי בערכים שנקבעו בפרמטר dwFlags. אם dwFlags נקבע לערך FORMAT_MESSAGE_FROM_HMODULE, אז lpSource הוא hModule של המודול שמכיל את טבלת ההודעות שצריך לחפש. אך אם קבעת את dwFlags ל-FORMAT_MESSAGE_FROM_STRING, אז lpSource הוא LPTSTR אשר מצביע לטקסט הודעה שאינו מפורמט. אם אינך קובע ב-dwFlags אף לא אחד משני דגלים אלה, הפונקציה מתעלמת מ- lpSource.
DwMessageId	ממזהה ההודעה בן 32 סיביות. הפונקציה מתעלמת מפרמטר זה אם dwFlags מכיל את הדגל FORMAT_MESSAGE_FROM_STRING.
dwLanguageId	מזהה השפה בן 32 סיביות עבור ההודעה הדרושה. הפונקציה מתעלמת מפרמטר זה אם dwFlags מכיל את הדגל FORMAT_MESSAGE_FROM_STRING. אם אתה מעביר LANGID מסוים בפרמטר זה, FormatMessage מחזירה הודעה רק עבור LANGID שצוין. אם הפונקציה אינה יכולה למצוא הודעה עבור LANGID זה, היא מחזירה ERROR_RESOURCE_LANG_NOT_FOUND.
lpBuffer	מצביע לחוצץ עבור ההודעה המפורמטת ו-NULL מסיים. אם dwFlags מכיל FORMAT_MESSAGE_ALLOCATE_BUFFER, הפונקציה מקצה חוצץ באמצעות הפונקציה LocalAlloc, ומציבה את כתובת החוצץ בכתובת שמוגדרת ב-lpBuffer.
nSize	אם אינך קובע את הדגל FORMAT_MESSAGE_ALLOCATE_BUFFER, פרמטר זה מציין את מספר הבתים המקסימלי (גרסת ANSI), או את מספר התווים (גרסת Unicode) אשר התוכנית יכולה לאחסן בחוצץ הפלט. אם אתה מגדיר FORMAT_MESSAGE_ALLOCATE_BUFFER, פרמטר זה מציין את מספר הבתים המינימלי, או את מספר התווים המינימלי, בהתאמה, שצריך להקצות לחוצץ הפלט.
Arguments	מצביע למערך של ערכים בני 32 סיביות שמשמשים כערכי כניסה (Insert values) בהודעה הערוכה. 1% בפורמט המחרוזת מציין את הערך הראשון במערך הארגומנטים ; 2% מציין את הארגומנט השני ; וכך הלאה.

הפונקציה מפרשת את הערך 32 סיביות לפי מידע העריכה שמוכל בפרמטר dwFlags והמקום הממשי של הגדרת ההודעה. ברירת המחדל היא להתייחס לכל ערך כמצביע למחרוזת המסתיימת ב-NULL. לפי ברירת המחדל, הפרמטר Arguments הוא מסוג va_list*, שהינו סוג נתונים המיושם במיוחד בשפה, כדי לטפל במספר ארגומנטים משתנה. אם אין מצביע מסוג va_list*, צריך להגדיר את הדגל FORMAT_MESSAGE_ARGUMENT_ARRAY ולהעביר מצביע למערך שמכיל ערכים בני 32 סיביות; ערכים אלה הם קלט לאיתור האיבר המתאים (הכניסה בטבלה) לקבלת ההודעה הערוכה. לכל ערך כניסה חייב להיות ערך מתאים במערך. טבלה 17.27 מפרטת את דגלי העריכה עבור הפרמטר dwFlags.

טבלה 17.27: הערכים האפשריים עבור הפרמטר **dwFlags**.

ערך	פירוש
FORMAT_MESSAGE_ALLOCATE_BUFFER	הפרמטר lpBuffer הוא מצביע למצביע מסוג PVOID, והפרמטר nSize מציין את מספר הבתים המינימלי (גרסת ANSI) או את מספר התווים המינימלי (גרסת Unicode) שצריך להקצות לחוצץ הפלט של ההודעה. הפונקציה מקצה חוצץ מספיק גדול, כדי שיכיל את ההודעה הערוכה, ומציבה מצביע לחוצץ שהוקצה בכתובת שמוגדרת על ידי lpBuffer.
FORMAT_MESSAGE_IGNORE_INSERTS	הפונקציה חייבת להתעלם מרצף כניסות (Insert Sequences) בהגדרת ההודעה, ולהעביר אותן אל חוצץ הפלט ללא שינוי. דגל זה שימושי לקבלת הודעה ועריכתה יותר מאוחר. אם אתה מגדיר דגל זה, הפונקציה מתעלמת מהפרמטר Arguments.
FORMAT_MESSAGE_FROM_STRING	מציין ש-lpSource הוא מצביע להגדרת הודעה המסתיימת ב-NULL. הגדרת ההודעה יכולה להכיל רצף כניסות, כמו שטקסט ההודעה שבמשאב טבלת ההודעות יכול להכיל רצף כניסות. אינך יכול להשתמש בדגל עריכה זה עם FORMAT_MESSAGE_FROM_HMODULE או עם FORMAT_MESSAGE_FROM_SYSTEM.

ערך	פירוש
FORMAT_MESSAGE_FROM_HMODULE	מגדיר ש-lpSource הוא ידית מודול שמכיל את טבלת ההודעות לחיפוש. אם הידית lpSource שווה ל-NULL, הפונקציה מחפשת בעותק הלוגי של התהליך הנוכחי של היישום. אינך יכול להשתמש בדגל עריכה זה עם FORMAT_MESSAGE_FROM_STRING.
FORMAT_MESSAGE_FROM_SYSTEM	מגדיר שהפונקציה צריכה לחפש בטבלת ההודעות של המערכת עבור ההודעה הדרושה. אם אתה מגדיר דגל זה עם FORMAT_MESSAGE_FROM_HMODULE, הפונקציה מחפשת בטבלת ההודעות של המערכת, אם אינה מוצאת את ההודעה במודול שמוגדר על ידי lpSource. אינך יכול להשתמש בדגל עריכה זה עם FORMAT_MESSAGE_FROM_STRING.
FORMAT_MESSAGE_ARGUMENT_ARRAY	מציין שהפרמטר Arguments **אינו** מבנה va_list*, אלא רק מצביע למערך של ערכים בני 32 סיביות שמייצגים את הארגומנטים.

אם הפונקציה מצליחה, היא מחזירה את מספר הבתים (גרסת ANSI) או את מספר התווים (גרסת Unicode) שמאוחסנים בחוצץ הפלט, ללא תו NULL המסיים; אם הפונקציה נכשלת, היא מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל- GetLastError.

בטקסט ההודעה, הפונקציה תומכת במספר רצפי חילוף (Escape Sequences) עבור עריכת הודעה דינמית. טבלה 17.28 מציגה רצפי חילוף אלה ואת הפירוש שלהם. כל רצפי החילוף מתחילים בתו "אחוז" (%).

טבלה 17.28: רצפי החילוף האפשריים של תווי הבקרה לעריכה עם התחילית %.

רצף החילוף	פירוש
%0	מסיים שורת טקסט הודעה מבלי להוסיף תו שורה חדשה בסופה. אפשר להשתמש ברצף חילוף זה כדי לבנות שורות ארוכות, או לסיים את ההודעה עצמה מבלי להוסיף תו שורה חדשה בסופה. דבר זה שימושי להודעות של בקשת אישור מהמשתמש.

רצף החילוף	פירוש
%n!printf format string!	מזהה כניסה. הערך של n יכול להיות בתחום מ- 1 עד 99. פורמט מחרוזת printf (המחרוזת חייבת להיות בין שני סימני קריאה) הוא רשות וברירת המחדל היא !s! אם אינך מגדיר אותה. פורמט מחרוזת printf יכול להכיל את תו הבקרה "*" בשביל הדיוק או רוחב הרכיב. אם אתה מגדיר "*" עבור מרכיב אחד, הפונקציה FormatMessage משתמשת בכניסה %n+1; היא משתמשת ב- %n+2 אם אתה מגדיר את "*" עבור שני המרכיבים. הפונקציה אינה תומכת בפורמט ה**נקודה הצפה** (Floating-Point) של printf עבור תווי הבקרה e, E, f, ו- g, של printf. החלופה היא להשתמש בפונקציה sprintf כדי לפרמט את המספר בשיטת הנקודה הצפה בחוצץ זמני, ואחר כל להשתמש בחוצץ זה כמחרוזת הכניסה.

הפונקציה עורכת כל תו אחר שאינו תו מספר, שמופיע אחרי סימן האחוז בפלט ההודעה. היא מציגה את התו עצמו, בלי סימן האחוז התחילי. טבלה 17.29 מציגה מספר דוגמאות לפלט תווים שאינם משמשים לפעולות עריכה.

טבלה 17.29: דוגמת פלט לתווים שאינם משמשים לפעולות עריכה (פירמוט).

פורמט מחרוזת	תוצאת הפלט
%%	סימן אחוז יחיד בטקסט ההודעה הערוכה.
%n	מעבר שורה קשיח כאשר פורמט המחרוזת נמצא בסוף שורה. פורמט מחרוזת זה שימושי כאשר FormatMessage מספקת מעברי שורה רגילים, כך שההודעה מתאימה לרוחב מסוים.
%space	רווח בטקסט ההודעה הערוך. תוכל להשתמש בעריכה זו של המחרוזת, כדי להציב מספר רווחים בסוף שורה של טקסט ההודעה הערוך.
%.	נקודה יחידה בטקסט ההודעה הערוך. תוכל להשתמש בפורמט מחרוזת זה, כדי להוסיף נקודה אחת בתחילת שורה, מבלי לסיים את הגדרת טקסט ההודעה.
%!	סימן קריאה יחיד בטקסט ההודעה הערוך. תוכל להשתמש בפורמט מחרוזת זה כדי להוסיף סימן קריאה מיד לאחר הטקסט, מבלי שתהיה מפורשת באופן שגוי כתחילת מחרוזת עריכה של printf.

17.44 קלט/פלט אסינכרוני עם אובייקט אירוע גרעין

בסעיף 17.41 השתמשת בפונקציה WaitForSingleObject יחד עם ידית התקן של התקן אסינכרוני כדי לבצע קלט/פלט אסינכרוני. השימוש באובייקט התקן גרעין, כמו שמוצג בסעיף 17.41, פשוט יחסית וברור למדי, אך הוא אינו שימושי לטיפול בדרישות קלט/פלט רבות בו-זמנית. אם, לדוגמה, אתה מנסה לבצע פעולות קלט/פלט אסינכרוני רבות כנגד קובץ יחיד בו-זמנית, ההמתנה עבור הידית אינה עוזרת לך, מכיון שהיא הופכת להיות מסומנת (Signaled) כאשר האירוע הראשון מסתיים, ועליך להמתין לה עוד פעם כדי שתשתחרר – דבר אשר יכול לגרום לה להמתין לצמיתות.

באפשרותך להשתמש גם בפונקציה CreateEvent כדי ליצור אובייקט אירוע גרעין. תוכל אז לזהות את האובייקט הזה על ידי האיבר hEvent של המבנה OVERLAPPED שמעביר אותו לפונקציית קלט/פלט אסינכרוני (ReadFile או WriteFile). כאשר מעבירים אירוע בצורה זו, מערכת ההפעלה קובעת את האירוע אוטומטית למסומן כאשר פעולת הקלט/פלט מסתיימת. אך מכיון שהתוכנית יכולה לקבוע אירוע שונה לכל פעולת קלט/פלט, היא גם יכולה להגיב לסיום פעולת קלט/פלט אחת ולא אחרת.

בכל פעם שאתה מבצע פעולת קלט/פלט אסינכרוני, התוכנית צריכה ליצור אירוע חדש עבור הפעולה הנדרשת. בצורה כזו, בכל פעם שמערכת ההפעלה מסיימת את העיבור המוטל עליה, היא קובעת את אירוע הפעולה הקוראת למצב מסומן. כמו שמתואר בסעיף 17.45, תוכל אז להמתין לאירועים שאתה רוצה לסיים.

17.45 WaitForMultipleObjects עם קלט/פלט אסינכרוני

למדת שבאפשרותך להשתמש בפונקציה WaitForMultipleObjects כדי להמתין לאירוע אחד מתוך אירועים רבים עד שיתרחש, או להמתין לאירועים של תת-קבוצה מסוימת שיתרחשו. כאשר אתה מבצע קלט/פלט אסינכרוני, צריך להשתמש ב- WaitForMultipleObjects כדי לסנכרן את המטלות שלך לקבוצה מסוימת של אירועים. התוכנית צריכה לקרוא ל- WaitForMultipleObjects עם הידיות של כל האירועים שמערכת ההפעלה צריכה לסיים לפני שהתוכנית תוכל להמשיך בעיבוד, ואז להמתין לאירועים אלה עד שיהיו במצב מסומן. בדרך כלל מבצעים עיבוד כזה על ידי שימוש בקוד שדומה לקטע הקוד שלהלן:

```
Event[1] = HANDLE CreateEvent(
                  LPSECURITY_ATTRIBUTES lpEventAttributes,
                  BOOL bManualReset, BOOL bInitialState,
                  LPCTSTR Event1);
Overlapped1.hEvent = Event[1];
ReadFile(hFile, bBuffer, sizeof(bBuffer), &dwNumBytesRead,
                  &Overlapped1);
```

```
Event[2] = HANDLE CreateEvent(
                LPSECURITY_ATTRIBUTES lpEventAttributes,
                BOOL bManualReset, BOOL bInitialState,
                LPCTSTR Event2);
Overlapped2.hEvent = Event[2];
ReadFile(hFile, bBuffer, sizeof(bBuffer), &dwNumBytesRead,
                &Overlapped2);
// Additional processing here
DWORD WaitForMultipleObjects(2, CONST HANDLE *Event,
                BOOL bWaitAll, INFINITE);
```

קטע הקוד יוצר אירוע ומעביר אותו לפעולת הקריאה הראשונה. אחר כך, קטע הקוד יוצר אירוע שני ומעביר אותו לפעולת הקריאה השנייה. לבסוף, הקוד ממתין לשני האירועים שיחזרו לפני שהוא ממשיך את העיבוד.

אובייקטי אירוע הגרעין שימושיים מאוד לניהול קלט/פלט אסינכרוני. הסיכון בשימוש באובייקטי אירוע גרעין הוא כאשר אתה קובע אובייקט אירוע גרעין לאירוע שמאופס אוטומטית (Auto-Reset Event), כי אפשר שהמטלה תיתקע לצמיתות בזמן שהיא ממתינה לאירוע auto-reset שיתאפס, גם אם הפונקציה כבר סיימה את פעולת הקלט/פלט. אם אתה קורא ל- GetOverlappedResult כדי לקבוע כמה בתים הועברו בהצלחה בפעולת הקלט/פלט, היא מאפסת את האירוע למצב אינו מסומן. בקיצור, צריך להשקיף מקרוב על סדר הפונקציות שמפעילים כאשר משתמשים באובייקטי גרעין, כדי לנהל קלט/פלט אסינכרוני בצורה נכונה.

17.46 יציאות קלט/פלט מסיימות (I/O Completion Ports)

הטכניקה הרביעית שתוכניות עשויות להשתמש בה לביצוע קלט/פלט אסינכרוני היא השימוש ב**יציאות סיום קלט/פלט** (I/O Completion Ports). בדרך כלל משתמשים ביציאות סיום קלט/פלט, כאשר בונים תוכנית שתשרת מאות, או אפילו אלפי משתמשים (כמו שרת Web). יציאות סיום קלט/פלט מאוד אמינות ויעילות, ויכולות לטפל ברמה טובה בנפחי תעבורה גדולים של פעולות תקשורת. כאשר יוצרים יישום שירות, עושים זאת בדרך כלל על ידי שימוש בשיטה אחת מבין שתים אלו:

- ✰**מודל טורי** (Serial Model): מטלה יחידה ממתינה לדרישה מהלקוח (בדרך כלל דרך הרשת). כאשר הדרישה מגיעה, המטלה מתעוררת ומטפלת בדרישה זו.
- ✰**מודל מקבילי** (Concurrent Model): מטלה יחידה ממתינה לדרישה מהלקוח ואחר כך יוצרת מטלה חדשה כדי שתטפל בדרישה. בזמן שהמטלה החדשה מטפלת בדרישת הלקוח, המטלה המקורית חוזרת למצב המתנה לדרישה נוספת מלקוח כלשהו. כאשר המטלה שמטפלת בדרישת הלקוח מסיימת את העיבוד המוטל עליה, היא נסגרת.

המודל הטורי מוגבל מאוד, בכך שאינו מטפל בדרישות רבות בו-זמנית, מפני שרק מטלה אחת מטפלת בדרישות. בניגוד לכך, המודל המקבילי יכול לטפל במספר גדול מאוד של דרישות המגיעות בו-זמנית, מפני שכל דרישה מקבלת מטלה נפרדת שמעבדת אותה. כאשר מתכננים שירותי Windows NT, התוכניות ישתמשו בדרך כלל במודל המקבילי.

הדרכה ליצירת שירות המבוסס על המודל המקבילי אינה כלולה בספר זה, אך לפניך מידע מספיק, כדי להבין את ההבדל בין שני סוגי השירות המיוצגים על ידי שני המודלים.

✍ **הערה:** תוכל להשתמש ביציאות סיום קלט/פלט רק תחת Windows NT. ל-Windows 9x אין את הפונקציונליות הדרושה למימוש שיטת קלט/פלט זו.

17.47 התרעות קלט/פלט (Alertable I/O) בעיבוד אסינכרוני

למדת שבכל פעם שפונקציה יוצרת מטלה, המערכת יוצרת עבורה גם תור הודעות, אשר מקושר אליה. מערכת ההפעלה יוצר תור נוסף ונפרד עבור המטלה שנוצרה, אשר ידוע בשם תור **קריאה לפרוצדורה אסינכרונית** - APC (Asynchronous Procedure Call). מערכת ההפעלה משתמשת ב**פונקציות בסיסית** (Low-Level Functions) של הגרעין כדי ליצור ולהחזיק את תור APC. מכיון שמערכת ההפעלה משתמשת בפונקציות גרעין ברמה בסיסית כדי להחזיק בתור APC, תור זה מהיר מאוד ומהווה שיטה יעילה לניהול קלט/פלט אסינכרוני.

באפשרות התוכניות לדרוש פעולות קלט/פלט באמצעות פונקציות שיעבירו את תוצאות דרישות הקלט/פלט ישירות אל תור APC של המטלה הקוראת. כדי לשלוח דרישות קלט/פלט שהסתיימו אל תור APC של המטלה, צריך להשתמש בפונקציות ReadFileEx ו- WriteFileEx, כמו שרואים בדוגמה זו:

```
BOOL ReadFileEx(HANDLE hFile, LPVOID lpBuffer,
                          DWORD nNumberOfBytesToRead,
                          LPOVERLAPPED lpOverlapped,
                          LPOVERLAPPED_COMPLETION_ROUTINE
                             lpCompletionRoutine );
BOOL WriteFileEx(HANDLE hFile, LPCVOID lpBuffer,
                          DWORD nNumberOfBytesToWrite,
                          LPOVERLAPPED lpOverlapped,
                          LPOVERLAPPED_COMPLETION_ROUTINE
                             lpCompletionRoutine );
```

כפי שניתן לראות, שתי הפונקציות מקבלות בפרמטר האחרון שלהן את הכתובת של **שיגרה מסיימת** (Complition Routine), כדי שתבוצע כאשר הן מסיימות את העיבוד שלהן. סעיף 17.49 מסביר את שתי הפונקציות ReadFileEx ו- WriteFileEx בפירוט. חייבים להשתמש בהגדרה הבאה עבור השיגרה המסיימת ששתי הפונקציות משתמשות בה:

```
VOID WINAPI FileIOCompletionRoutine(
  DWORD dwErrorCode,                // completion code
  DWORD dwNumberOfBytesTransfered,  // number of bytes
                                    // transfered
  LPOVERLAPPED lpOverlapped         // pointer to structure
                                    // with I/O information
);
```

הפרמטר dwErrorCode מגדיר את מצב סיום הקלט/פלט. הפרמטר dwErrorCode יכול להיות אחד הערכים שמפורטים בטבלה 17.30.

טבלה 17.30: הערכים האפשריים של הפרמטר **dwErrorCode**.

ערך	פירוש
0	פעולת קלט/פלט הסתיימה בהצלחה.
ERROR_HANDLE_EOF	הפונקציה ניסתה לקרוא מעבר לסוף הקובץ.

הפרמטר dwNumberOfBytesTransfered מגדיר את מספר הבתים שהועברו. אם קורית שגיאה, פרמטר זה שווה לאפס. הפרמטר lpOverlapped מצביע למבנה OVERLAPPED שמוגדר על ידי פונקציית קלט/פלט אסינכרוני. Windows אינה משתמשת באיבר hEvent של המבנה OVERLAPPED; היישום הקורא יכול להשתמש באיבר זה כדי להעביר מידע לשיגרה המסיימת. Windows אינה משתמשת במבנה OVERLAPPED לאחר שהתוכנית קוראת לשיגרה המסיימת, כך ששיגרה זו יכולה להקצות מחדש את הזיכרון ששימש את המבנה OVERLAPPED.

הפונקציה FileIOCompletionRoutine היא למעשה מחזיק מקום (Placeholder) עבור שם פונקציה שמוגדר על ידי היישום או על ידי הספריה. החזרה מהפונקציה FileIOCompletionRoutine מאפשרת ל-Windows לקרוא לשיגרה מסיימת אחרת לביצוע פעולת קלט/פלט. כל השגרות המסיימות הממתינות נקראות לפני שמסתיימת ההמתנה של **המטלה שמצפה להתראה** (Alertable Thread), כאשר מתקבל הקוד המוחזר WAIT_IO_COMPLETION. יכולה להיות ש-Windows תקרא לשגרות מסיימות הממתינות בסדר כלשהו, וגם ייתכן שהיא תקרא להן לפי הסדר שבו התוכנית סיימה את פונקציות הקלט/פלט, או שלא תקרא להן בסדר זה. בכל פעם ש-Windows קוראת לשיגרה מסיימת, היא משתמשת בחלק מסוים ממחסנית התהליך. אם השיגרה מבצעת קלט/פלט אסינכרוני ויש הודעות ממתינות, המחסנית יכולה לגדול.

17.48 התרעות קלט/פלט (Alertable I/O) פועלות רק תחת Windows NT

למדת שהתרעת קלט/פלט היא טכניקה מתקדמת לטיפול בקלט/פלט אסינכרוני אשר משתמשת בתור הודעות קלט/פלט ובפונקציית משוב (Callback Function) אחת או יותר. מכיון שטכניקת התרעות קלט/פלט משתמשת בגירסה המורחבת של הפונקציות ReadFile ו- WriteFile, התוכנית יכולה להשתמש בה רק אם אתה בטוח שהיא תופעל תחת מערכות Windows NT בלבד. אם אתה מנסה להשתמש ב-ReadFileEx או WriteFileEx ב- Windows 9x או במערכות Win32, הפונקציות מחזירות False ואינן מבצעות פעולה כלשהי. קריאה ל- GetLastError מחזירה ERROR_CALL_NOT_IMPLEMENTED. אל תנסה להשתמש בטכניקה זו ב- Windows 9x, מכיון שהיא תגרום לתוצאות שאין לצפותן מראש.

17.49 הפונקציות ReadFileEx ו-WriteFileEx

למדת בסעיף 17.45 שהתוכניות יכולות להשתמש בפונקציות ReadFileEx ו-WriteFileEx כדי לבצע קלט/פלט אסינכרוני תחת Windows NT. הפונקציה ReadFileEx קוראת נתונים מקובץ בצורה אסינכרונית. מתכנתים משתמשים ב-ReadFileEx רק בפעולה אסינכרונית, ולא כמו בפונקציה ReadFile, אשר משתמשים בה הן עבור פעולה סינכרונית והן עבור פעולה אסינכרונית. ReadFileEx מאפשרת ליישום לבצע טיפול אחר במהלך פעולת קריאת הקובץ. הפונקציה ReadFileEx מדווחת על **מצב הסיום** (Completion Status) שלה באופן אסינכרוני, וקוראת לרוטינה מסיימת שאתה מגדיר כאשר היא מסיימת לקרוא והמטלה הקוראת **במצב המתנה להתרעה** (Alertable Wait State). משתמשים בפונקציה ReadFileEx כמו שרואים בהגדרה שלהלן:

```
BOOL ReadFileEx(
  HANDLE hFile,                    // handle of file to read
  LPVOID lpBuffer,                 // address of buffer
  DWORD nNumberOfBytesToRead,      // number of bytes to read
  LPOVERLAPPED lpOverlapped,       // address of offset
  LPOVERLAPPED_COMPLETION_ROUTINE
  lpCompletionRoutine       // address of completion routine
);
```

הפונקציה ReadFileEx מקבלת את הפרמטרים שמפורטים בטבלה 17.31.

כאשר הפונקציה ReadFileEx מצליחה, הערך המוחזר שונה מאפס; ואם הפונקציה נכשלת, היא מחזירה אפס. כדי לקבל מידע שגיאה מורחב, צריך לקרוא ל-GetLastError. כאשר הפונקציה מצליחה, יש למטלה הקוראת פעולת קלט/פלט

אסינכרוני לביצוע: פעולת הקריאה החופפת (Overlapped) מהקובץ. כאשר פעולת הקריאה החופפת מסתיימת והמערכת חוסמת את המטלה הקוראת במצב המתנה להתרעה, המערכת קוראת לפונקציה שמוצבעת על ידי lpCompletionRoutine, ומצב ההמתנה מסתיים עם קוד מוחזר ששווה ל- WAIT_IO_COMPLETION.

טבלה 17.31: הפרמטרים שמקבלת הפונקציה **ReadFileEx**.

פרמטר	תיאור
hFile	ידית פתוחה שמגדירה את ישות הקובץ שצריך לקרוא ממנו. חייבים ליצור ידית קובץ זו עם הדגל FILE_FLAG_OVERLAPPED והיא חייבת להיות בעלת גישה GENERIC_READ לקובץ. הפרמטר hFile יכול להיות ידית כלשהי, אשר נפתחה על ידי הפונקציה CreateFile עם הדגל FILE_FLAG_OVERLAPPED.
lpBuffer	מצביע לחוצץ שמקבל את הנתונים שהפונקציה קוראת מהקובץ. היישום אינו צריך להשתמש בחוצץ זה, עד שהפונקציה מסיימת את הקריאה.
nNumberOfBytesToRead	מספר הבתים שהפונקציה צריכה לקרוא מהקובץ.
lpOverlapped	מצביע למבנה מסוג OVERLAPPED שמספק נתונים לפונקציה, כדי להשתמש בהם במהלך הקריאה האסינכרונית. אם הקובץ שמוגדר על ידי hFile תומך בהיסט בית (Byte Offset), התהליך הקורא של ReadFileEx חייב להגדיר היסט בית בקובץ שממנו מתחילים לקרוא. התהליך הקורא מגדיר את היסט הבית על ידי קביעת הערכים המתאימים לאיברים Offset ו-OffsetHeigh של המבנה OVERLAPPED. אם ישות הקובץ שמוגדרת על ידי hFile אינה תומכת בהיסט בית (לדוגמה, אם זה צינור בעל שם), התהליך הקורא חייב לקבוע את האיברים Offset ו-OffsetHeigh לאפס, או שהפונקציה ReadFileEx נכשלת. הפונקציה ReadFileEx מתעלמת מהאיבר hEvent של המבנה OVERLAPPED. הפונקציה ReadFileEx מסמנת שסיימה את הקריאה על ידי קריאה לשיגרה מסיימת שמוצבעת על ידי lpCompletionRoutine, או על ידי משלוח הודעה לשיגרה זו, ולכן היא אינה צריכה ידית אירוע. הפונקציה ReadFileEx משתמשת באיברים Internal ו-InternalHeigh של המבנה OVERLAPPED. היישום אינו צריך לקבוע איברים אלה. המבנה OVERLAPPED שמוצבע על ידי lpOverlapped חייב להישאר תקף במהלך פעולת הקריאה.

פרמטר	תיאור
lpCompletionRoutine	מצביע לרוטינה המסיימת ש-Windows צריכה לקרוא לה, כשפעולת הקריאה מסתיימת והמטלה הקוראת **במצב המתנה להתרעה** (Alertable Wait State).

כאשר הפונקציה מצליחה ופעולת הקריאה מהקובץ מסתיימת, אבל המטלה הקוראת אינה במצב המתנה להתרעה, המערכת מכניסה לתור את הקריאה לפונקציה המסיימת, ומחזיקה בקריאה עד שהמטלה הקוראת נכנסת למצב המתנה להתרעה. אם ReadFileEx מנסה לקרוא מעבר לסוף הקובץ, הפונקציה מחזירה אפס, ו-GetLastError מחזירה ERROR_HANDLE_EOF.

אם תהליך אחר נועל חלק מהקובץ שמוגדר על ידי hFile, ופעולת הקריאה שמוגדרת בקריאה ל- ReadFileEx חופפת לחלק שננעל, הקריאה ל- ReadFileEx נכשלת. אם ReadFileEx מנסה לקרוא נתונים **מחריץ דואר** (Mailslot) שהחוצץ שלו קטן מדי, הפונקציה מחזירה False, ו-GetLastError מחזירה ERROR_INSUFFICIENT_BUFFER. אסור שהיישומים יקראו מחוצץ הקלט וגם אסור להם לכתוב בחוצץ הקלט שפעולת הקריאה משתמשת בו, עד שפעולת הקריאה מסתיימת. גישה לפני הזמן לחוצץ הקלט יכולה לגרום להשחתת הנתונים שהפונקציה קוראת לחוצץ זה. הפונקציה ReadFileEx יכולה להיכשל אם יש יותר מדי דרישות קלט/פלט אסינכרוני שממתינות לביצוע. במקרה של כישלון, GetLastError יכולה להחזיר את ההודעה ERROR_INVALID_USER_BUFFER או את ההודעה ERROR_NOT_ENOUGH_MEMORY (כמו שלמדת בסעיף 17.39).

אם מנסים לקרוא מכונן דיסקטים שאין בו דיסקט, המערכת מציגה תיבת הודעה שמבקשת לאשר את הפעולה "נסה שנית". כדי למנוע מהמערכת להציג תיבת הודעה זו, צריך לקרוא לפונקציה SetErrorMode עם SEM_NOOPENFILEERRORBOX. אם הפרמטר hFile מכיל ידית של צינור בעל שם או ישות קובץ אחר שאינו תומך בהיסט בית, האיברים Offset ו-OffsetHeigh של המבנה OVERLAPPED שמוצבע על ידי lpOverlapped חייבים להיות אפס, או ש-ReadFileEx נכשלת.

17.50 שגרת משוב מסיימת (CALLBACK Completion Routine)

למדת בסעיף 17.48 שהתוכניות חייבות להשתמש בשיגרת משוב מסיימת עם שתי הפונקציות ReadFileEx ו-WriteFileEx. עליך להשתמש בהגדרה שלהלן עבור השיגרה המסיימת שנקראת על ידי שתי הפונקציות האלו.

```
VOID WINAPI FileIOCompletionRoutine(
  DWORD dwErrorCode,                  // completion code
  DWORD dwNumberOfBytesTransfered,    // number of bytes
                                      // transfered
```

```
  LPOVERLAPPED lpOverlapped           // pointer to structure
                                      // with I/O information
);
```

כאשר קוראים לאחד **מאובייקטי ההמתנה** (Wait Objects) ומציבים את המטלה **במצב התרעה** (Alertable State), מערכת ההפעלה בודקת תחילה את תור ה**קריאה לפרוצדורה אסינכרונית** - Asynchronous Procedure Call) APC). אם יש לפחות כניסה אחת בתור, המערכת אינה **מרדימה** (Sleep) את המטלה; במקום זאת, היא מקבלת את האיבר מתור APC והמטלה שלך קוראת לשיגרת המשוב, ומעבירה לה את קוד השגיאה של פעולת הקלט/פלט שהסתיימה, את מספר הבתים שהועברו, ואת כתובת המבנה OVERLAPPED שהמטלה מסרה בעת הצגת הדרישה לפעולת קלט/פלט. לאחר **ששיגרת המשוב** מבצעת את העיבוד, המערכת בודקת אם יש איברים נוספים בתור APC. אם יש איברים נוספים, היא מעבירה אותם לשיגרת המשוב לפי סדר. אם התור ריק, **הפונקציה המתריעה** (Alertable Function) חוזרת, והמטלה ממשיכה בעיבוד מבלי להירדם. לכן, הזמן היחיד שהמטלה עלולה להירדם בו הוא כאשר התור ריק.

17.51 תוכנית קלט/פלט מתריעה (Alertable I/O Program)

למדת שבאפשרות התוכניות להשתמש בטכניקות רבות העוצמה של **התרעות קלט/פלט** (Alertable I/O) כדי לבצע פעולות קלט ופלט אסינכרוני מורכבות בקבצים. התקליטור שמצורף לספר זה מכיל את התוכנית **Alertable_IO**, אשר משתמשת בהתרעות קלט/פלט לביצוע משימת העתקה פשוטה אחת. כאשר מהדרים ומפעילים את התוכנית, היא תשתמש בהתרעת קלט/פלט כדי להעתיק קובץ, ותודיע על העיבוד וסיום הפעולה.

כאשר התוכנית מתחילה, היא יוצרת קבוצה של דרישות קלט/פלט. כדי לעשות זאת, היא מאתחלת קבוצת מבנים מסוג MAX_PENDING_IO_REQS, שהיא משתמשת בהם כדי להודיע למערכת ההפעלה על המספר הגדול ביותר האפשרי של דרישות קלט/פלט שישנן בו-זמנית. כל מבנה מכיל מבנה OVERLAPPED, ולכן אין מבנה כלשהו שמכיל מצביע באיבר hEvent. בנוסף למבנה OVERLAPPED שדרוש לכל בקשת קלט/פלט, כל בקשה כזו גם צריכה חוצץ בזיכרון שמוחזק במבנה IO_REQS.

אחרי שהתוכנית מאתחלת את המבנה IO_REQS, היא קוראת לפונקציה ReadFileEx כדי לדרוש ממערכת ההפעלה לקרוא מהקובץ. בנקודה זו, התוכנית מתחילה להשתמש בהתרעות קלט/פלט. התהליך חוזר מיד לתיבת הדו-שיח, והמשתמש יכול להתחיל מיד בהעתקת קובץ נוסף. עם זאת, ברקע ReadFileEx מוצאת וקוראת את הקובץ. כאשר היא מסיימת את הפעולה, היא מודיעה לתהליך (באמצעות שגרת המשוב) שהיא משתמשת בנתונים שקראה קודם בכדי לכתוב את העתק הקובץ. למרות שהתוכנית ארוכה מאוד מלרשום אותה כולה כאן, כדאי לנתח את שתי פונקציות המשוב בלבד.

```
void WINAPI ReadCompletionRoutine(DWORD dwErrorCode,
                        DWORD dwNumberOfBytesTransfered,
                        LPOVERLAPPED lpOverlapped)
{
  PIOREQ pIOReq = (PIOREQ) lpOverlapped;
  chASSERT(dwErrorCode == NO_ERROR);
  g_cs.nReadsInProgress--;
  // Round up the number of bytes
  // to write to a sector boundary
  dwNumberOfBytesTransfered = (dwNumberOfBytesTransfered +
                        g_cs.dwPageSize-1) & ~(g_cs.dwPageSize-1);
  chVERIFY(WriteFileEx(g_cs.hFileDst, pIOReq->pbData,
                        dwNumberOfBytesTransfered,
                        lpOverlapped, WriteCompletionRoutine));
  g_cs.nWritesInProgress++;
}

void WINAPI WriteCompletionRoutine(DWORD dwErrorCode,
                        DWORD dwNumberOfBytesTransfered,
                        LPOVERLAPPED lpOverlapped)
{
  PIOREQ pIOReq = (PIOREQ) lpOverlapped;
  chASSERT(dwErrorCode == NO_ERROR);
  g_cs.nWritesInProgress--;

  if (g_cs.ulNextReadOffset.Quadpart <
                        g_cs.ulFileSize.Quadpart)
    {
      // the function hase't read past the end of file yet
      // so, read the next chunk of data
      lpOverlapped->Offset = g_cs.ulNextReadOffset.LowPart;
      lpOverlapped->OffsetHigh =
                        g_cs.ulNextReadOffset.HighPart;
      chVERIFY(ReadFileEx(g_cs.hFileSrc, pIOReq->pbData,
                BUFFSIZE, lpOverlapped,
ReadCompletionRoutine));
      g_cs.nReadsInProgress++;
      g_cs.ulNextReadOffset.Quadpart += BUFFSIZE;
    }
}
```

כמו שאפשר לראות, פונקציית המשוב ReadCompletionRoutine קוראת לפונקציה WriteFileEx עם המידע שמוחזר על ידי ReadFileEx ועם כתובת פונקציית המשוב WriteCompletionRoutine. השיגרה WriteCompletionRoutine בודקת את הגודל הנוכחי של הקובץ המועתק בכל פעם שהפונקציה WriteFileEx חוזרת; אם ההעתקה לא הסתיימה עדיין, השיגרה WriteCompletionRoutine משנה את מקום ההיסט (offset location) שבקובץ וקוראת לפונקציה ReadFileEx עוד פעם. אם ההעתקה הסתיימה, הפונקציה יוצאת והתוכנית מטפלת בניקוי הקובץ במקום אחר. כמו שאפשר לראות, מימוש התרעות קלט/פלט בתוכניות קל יחסית, ועל כן שיטה זו מאוד שימושית לביצוע קלט/פלט אסינכרוני.

✍ **הערה:** זכור! התוכנית **Alertable_IO** פועלת בצורה נכונה רק תחת מערכת Windows NT.

נספח א -
פונקציות נוספות והרחבה

הפונקציה ShowWindow

פונקציה זו קובעת את מצב התצוגה של החלון. היא מיועדת להציג חלונות על המסך.

```
BOOL ShowWindow(HWND hWnd, int nCmdShow);
```

הפונקציה ShowWindow מחזירה ערך בוליאני true או false. ערך זה מכיל את מצב התצוגה הקודם של החלון.

הערך יהיה true כאשר החלון במצבו הקודם היה מוצג (Visible).

הערך יהיה false כאשר החלון במצבו הקודם היה בלתי מוצג(Invisible).

הפרמטר hWnd הוא ידית החלון שברצונך להציג. הפרמטר hWnd שבתוכנית **generic**, מתייחס לחלון החדש שנוצר (**ראה פרק 2**). מצב התצוגה של החלון נקבע בפרמטר nCmdShow. הערך של nCmdShow חייב להיות אחד מהערכים שמפורטים בטבלה 1 שלפניך.

טבלה 1: הערכים עבור הפרמטר **nCmdShow**.

הערך	הפעולה
SW_FORCEMINIMIZE רק ל NT 5.0.	הקטנת החלון אפי' כאשר המטלה שלה הוא שייך תקועה. יש להשתמש בערך זה רק כאשר מקטינים חלונות ממטלה (Thread) אחרת.
SW_HIDE	הסתרת החלון.
SW_MINIMIZE	הקטנת החלון לגודל מינימלי (סמל) והפעלת החלון הבא שבציר z .
SW_MAXIMIZE	מגדיל את החלון לגודל המקסימלי.

הערך	הפעולה
SW_RESTORE	הפעלת החלון והצגתו. אם החלון הוקטן לסמל או נמצא בגודל מקסימלי, ShowWindow מחזירה את החלון לגודלו ומיקומו המקוריים.
SW_SHOW	קובע את החלון כחלון פעיל ומציג אותו בגודלו ומיקומו הנוכחיים.
SW_SHOWDEFAULT	הצגת החלון במצב ברירת המחדל של היישום. ShowWindow משיגה את מצב ברירת המחדל של היישום ממבנה STARTUPINFO שנלמד עליו בסעיפים הבאים.
SW_SHOWMAXIMIZED	מפעיל את החלון ומציג אותו בגודל מקסימלי.
SW_SHOWMINIIMIZED	קובע אותו כחלון הפעיל ומציגובגודל מינימלי, כסמל.
SW_SHOWMINNOACTIVE	הצגת החלון זה במצב מינימלי, והחלון הפעיל כרגע נשאר במצבו.
SW_SHOWNA	הצגת החלון במצב הנוכחי שלו, והחלון הפעיל נשאר במצבו.
SW_SHOWNOACTIVE	הצגת החלון בגודל ובמקום העדכניים ביותר. החלון הפעיל נשאר החלון הפעיל.
SW_SHOWNORMAL	קובע אותו כחלון הפעיל ומציגו בגודלו הרגיל.

בפעם הראשונה שהחלון מוצג על המסך, עליך להעביר את הפרמטר nWinMode בפונקציה ( )WinMain כפרמטר nHow.

השינויים הכלולים בתוכנית **tst_max** שלהלן עבור **generic**, מכוונים להודיע ל-Windows להגדיל את חלון היישום לגודל מקסימלי כשהמשתמש בוחר באפשרות Test מתוך תפריט File. מחק את הקוד הנוכחי בפונקציה WndProc, והחלף אותו בקוד הזה:

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg,
WPARAM wParam, LPARAM lParam)
 {
   switch(uMsg)
    {
      case WM_COMMAND:
        switch(LOWORD(wParam))
          {
            case IDM_TEST :
               ShowWindow(hWnd, SW_SHOWMAXIMIZED);
               break;
```

```
            case IDM_EXIT :
                DestroyWindow(hWnd);
                break;
          }
        break;
    case WM_DESTROY :
        PostQuitMessage(0);
        break;
    default:
        return (DefWindowProc(hWnd, uMsg, wParam, lParam));
  }
  return(0L);
}
```

התוצאה היחידה לשינוי הקוד של התוכנית המקורית **generic** מתבטאת בכך שכרגע החלון מוגדל למקסימום ותופס את השטח של כל המסך כשהמשתמש בוחר באפשרות Test! מתוך תפריט החלון. לאחר שתהדר ותפעיל את הגירסה החדשה של התוכנית **generic**, נסה אותה תחילה על ידי הגדלת חלון התוכנית למקסימום, ואחר כך החזר את החלון לגודלו הרגיל. לצורך בדיקה של ערכים נוספים, בנה חלון שמקבל כפרמטר בצורה כלשהיא HWND - ידית לחלון אחר, וקרא לו למשל hSecWnd, ונסה לבדוק מה קורה כאשר אתה שם בתוך הפונקציה הזו ערכים אחרים עבור nCmdShow, ועבור הפרמטר הראשון אתה שם את hSecWnd.

WM_VSCROLL ,WM_HSCROLL

הודעות אלו מיועדות לטפל באירועי גלילה.

Windows שולחת את ההודעה WM_VSCROLL לחלון כאשר מתרחש אירוע גלילה בפס הגלילה האנכי הסטנדרטי של החלון. Windows גם שולחת את ההודעה WM_VSCROLL לבעלים של פקד פס גלילה האנכי, כאשר מתרחש אירוע של גלילה בפקד. כאשר התוכנית מקבלת את ההודעה WM_VSCROLL, מילת הסדר-הנמוך של הפרמטר wParam מגדירה ערך פס גלילה שמציין את דרישת המשתמש לגלול. מילת הסדר-הנמוך יכולה להיות אחד מרשימת הערכים שמפורטים בטבלה 2.

טבלה 2: הערכים האפשריים למילת הסדר-הנמוך של הפרמטר **wParam**.

ערך	פירוש
SB_BOTTOM	גלילה עד למטה ימינה.
SB_ENDSCROLL	סיום הגלילה.
SB_LINEDOWN	גלילה של שורה אחת למטה.

ערך	פירוש
SB_LINEUP	גלילה של שורה אחת למעלה.
SB_PAGEDOWN	גלילה של דף אחד למטה.
SB_PAGEUP	גלילה של דף אחד למעלה.
SB_THUMBPOSITION	גלילה למקום מוגדר. זהו ערך מספרי מדויק של **היסט** (Offset) מתחילת החלון, לדוגמה: "12 שורות למטה מגבול עליון". הפרמטר nPos מגדיר את המקום הנוכחי.
SB_THUMBTRACK	גרירת תיבת הגלילה למקום מסוים. הפרמטר nPos מגדיר את המיקום הנוכחי.
SB_TOP	גלילה עד למעלה שמאלה.

בנוסף לערך ש-Windows מעבירה במילת הסדר-הנמוך של הפרמטר wParam, צריך לבדוק גם את מילת הסדר-הגבוה של wParam. מילת הסדר-הגבוה מציינת את המיקום הנוכחי של תיבת הגלילה, כאשר הפרמטר nScrollCode שווה ל-SB_THUMBPOSITION או ל-SB_THUMBTRACK; אחרת, הערך שמכילה מילת הסדר-הגבוה אינו משמעותי. לבסוף, הפרמטר lParam מכיל את ידית פס הגלילה. אם פס הגלילה לא שלח את ההודעה, ערכו של lParam יהיה NULL.

יישומים שמספקים משוב כאשר המשתמש גורר את תיבת הגלילה משתמשים באופן טיפוסי בהודעת שינוי המצב (Notification Message) SB_THUMBTRACK. אם יישום גולל את תוכן החלון, חובה על היישום להשתמש גם בפונקציה SetScrollPos כדי לקבוע מחדש את מיקום תיבת הגלילה.

Windows שולחת את ההודעה WM_HSCROLL לחלון כאשר מתרחש אירוע גלילה בפס הגלילה האופקי הסטנדרטי של החלון. Windows שולחת את ההודעה WM_HSCROLL גם לבעלים של פקד פס גלילה האופקי, כאשר מתרחש אירוע גלילה בפקד. כמו שההודעה WM_VSCROLL מכילה מידע נוסף בפרמטרים wParam ו-lParam, כך עושה גם ההודעה WM_HSCROLL.

עם WM_HSCROLL, מילת הסדר-הנמוך של wParam מגדירה ערך פס גלילה שמציין את דרישת המשתמש לגלול. מילת הסדר-הנמוך של wParam יכולה להכיל אחד מהערכים שמפורטים בטבלה 3.

טבלה 3: הערכים האפשריים למילת הסדר-הנמוך של הפרמטר **wParam**.

ערך	פירוש
SB_BOTTOM	גלילה עד למטה ימינה.
SB_ENDSCROLL	סיום הגלילה.

ערך	פירוש
SB_LINELEFT	גלילה שמאלה יחידה אחת.
SB_LINERIGHT	גלילה ימינה יחידה אחת.
SB_PAGELEFT	גלילה שמאלה על פי ערך רוחב החלון.
SB_PAGERIGHT	גלילה ימינה על פי ערך רוחב החלון.
SB_THUMBPOSITION	גלילה למקום מוגדר. זהו ערך מספרי מדויק של **היסט** (Offset) מצד שמאל של החלון. הפרמטר nPos (מילת הסדר-הגבוה של wParam) מגדיר את המיקום הנוכחי.
SB_THUMBTRACK	גרירת תיבת הגלילה למקום מסוים. הפרמטר nPos (מילת הסדר-הנמוך של wParam) מגדיר את המיקום הנוכחי.
SB_TOP	גלילה עד למעלה שמאלה.

כמו שהודעה WM_VSCROLL, מילת הסדר-הגבוה של wParam מגדירה את המיקום הנוכחי של תיבת הגלילה, כאשר הפרמטר nScrollCode שווה ל-SB_THUMBPOSITION או ל-SB_THUMBTRACK; אחרת, הערך שמכילה מילת הסדר-הגבוה לא משמעותי. הפרמטר lParam מחזיר את ידית פס הגלילה אם פס הגלילה שלח את ההודעה. אם פס הגלילה לא שלח את ההודעה, הערך של hwndScrollbar יהיה NULL.

שים לב, שתי ההודעות WM_HSCROLL ו-WM_VSCROLL מכילות ערך בן 16 סיביות של מקום תיבת הגלילה. לכן, ליישומים שמשתמשים רק ב-WM_HSCROLL וב-WM_VSCROLL כדי לקבוע את מיקום הגלילה של הנתונים יש למעשה ערך מקסימלי של 65,535. אך מכיון שהפונקציות SetScrollPos, SetScrollRange, GetScrollPos ו-GetScrollRange תומכות בערך בן 32 סיביות למיקום הגלילה של הנתונים, יש דרך להתגבר על מחסום 16 הסיביות של ההודעות WM_HSCROLL ו-WM_VSCROLL על ידי שימוש בפונקציות אלו. התבונן בעזרה הנלווית למהדר לקבלת מידע נוסף על הדרך להתגבר על מחסום 16 הסיביות.

פונקציות משוב (Callback Function)

במספר מקומות ראית שהוכרזו מספר פונקציות על ידי השימוש במילת המפתח CALLBACK. עליך להתייחס לפונקציה שהתוכנית שלך מכריזה (Declare) עליה באמצעות מילת המפתח CALLBACK כפונקציית משוב (Callback Function). פונקציה זו מעבירה לתוכנית כתובת של פונקציה שלישית, ואז פונקציה שלישית זו "קוראת חזרה" (Call Back) ומספקת מידע. תמיד תגדיר את הפונקציה WndProc כפונקציית callback. בתוכניות שתכתוב תשתמש פעמים רבות בפונקציות משוב יחד עם פונקציות API ספציפיות, כמו EnumFontFamilies ו-EnumWindows. כשתמסור את הכתובת של

פונקציית משוב לאחת מפונקציות אלו, הפונקציה תיקרא לפונקציית משוב עבור כל פריט שברשימה. לדוגמה, אם תיקרא ל-EnumWindows, תעביר לה לדוגמה את הכתובת של פונקציית משוב שמציגה ערכים או מוסיפה אותם למערך. אחר כך, EnumWindows תקרא לפונקציית משוב עבור כל חלון שברשימת החלונות שלה.

פונקציות מסוג callback דרושות ב-Windows, מכיון שהתוכניות שלך אמורות לטפל בפעולות התכנות הרבות והחשובות דרך **ממשק תכנות היישומים** (Application Program Interface - בקיצור API) של Windows, שהתוכנית שלך אינה יכולה לשנות באופן ישיר. לכן, אתה חייב לספק ל-API את האמצעים כדי להשתמש בהם לקריאה לשגרות שלך, בעת שהממשק מחזיר מידע מורחב (כמו לדוגמה, רשימה)...

הפונקציה LoadMenu

התוכניות משתמשות בפונקציה LoadMenu כדי לטעון תפריט שהוגדר קודם לכן בקובץ המשאבים. בדרך כלל תבצע אחד משני דברים: תקרא לפונקציה SetMenu אחרי הקריאה לפונקציה LoadMenu, או שתשתמש בקריאה לפונקציה LoadMenu מתוך הפונקציה CreateWindow. הפונקציה LoadMenu טוענת את משאב התפריט שמוגדר על ידי הפרמטר lpMenuName מקובץ ההפעלה (EXE File) ש-Windows קושרת אותו עם המופע של היישום (Application Instance). משתמשים בפונקציה LoadMenu בתוכניות, כמו בדוגמה שלהלן:

```
HMENU LoadMenu(
    HINSTANCE hInstance, // handle of application
                         // instance
    LPCTSTR lpMenuName   // menu name string or
                         // menu-resource identifier
);
```

LoadMenu מחזירה ידית (Handle) מסוג HMENU ומקבלת בפרמטרים שלה את ידית המופע של המודול שמכיל את משאב התפריט ש-LoadMenu צריכה לטעון. הפרמטר השני שהפונקציה LoadMenu מקבלת הוא מצביע למחרוזת המסתיימת ב-NULL שבו נמצא שם משאב התפריט. במקום להשתמש במצביע מחרוזת לשם התפריט, תוכל להשתמש ב-DWORD בפרמטר השני (המצביע לשם התפריט). במקרה כזה, הערך DWORD יהיה מורכב מ**מזהה משאב** (Resource Identifier) יחד עם המזהה האמיתי שבמילת הסדר-הנמוך, ואפס במילת הסדר-הגבוה. כדי ליצור את הערך של המילה הכפולה, השתמש במאקרו MAKEINTRESOURCE, ולא במצביע למחרוזת.

כדי להבין טוב יותר את הפעולה של הפונקציה LoadMenu, התבונן בקטע הקוד הבא (מהתוכנית **2_menus**. התוכנית נמצאת בתקליטור בתיקיה **Nis_A**), שמחליפה בין תפריט אחד לחברו, על פי בחירת המשתמש. שני הקבצים, קובץ המשאבים וקובץ התוכנית, שונים מהקבצים המקבילים להם בתוכנית **generic**. הקובץ **2_menus.rc** מכיל את הגדרות התפריט הבאות.

```
OLDMENU MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",    IDM_EXIT
    END
    MENUITEM "&New Menu!",  IDM_NEW
END
NEWMENU MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",    IDM_EXIT
    END
    MENUITEM "&Old Menu!",  IDM_OLD
END
```

ההצהרה הראשונה יוצרת את התפריט OLDMENU עם האפשרות !New Menu. כאשר המשתמש בוחר באפשרות !New Menu, התוכנית תחליף את התפריט OLDMENU שבחלון התוכנית לתפריט NEWMENU. כמו שאולי ציפית, הטיפול אשר משנה את התפריט של היישום מ-OLDMENU ל-NEWMENU נמצא בפונקציה WndProc של התוכנית **2_menus**.

התוכנית בודקת את הערך של הקבוע שמוכל במילת הסדר-הנמוך של הפרמטר wParam. על פי הערך המתקבל, התוכנית טוענת את התפריט האחר (במילים אחרות, אם התוצאה מראה שהתוכנית מציגה כרגע את המשאב OLDMENU, היא תחליף אותו ותציג את המשאב NEWMENU, ולהיפך). אחר כך, התוכנית משתמשת בפונקציה SetMenu לביצוע החלפות אלו. לבסוף, התוכנית קוראת לפונקציה DrawMenuBar אחרי שהיא מסיימת את ההחלפה, דבר המבטיח ש-Windows תצייר מחדש את התפריט שנטען במקום המיועד לו.

✍ **הערה:** התוכניות חייבות להשתמש בפונקציה **DestroyMenu** כדי שהיישום יפרק את התפריט וישחרר את הזיכרון שנתפס על ידי התפריט הקודם.

ממשק התכנות Win32 API מספק אוסף של פונקציות שאפשר להשתמש בהן בתוכניות, כדי לשנות תפריט בזמן פעולת היישום. ככלל, הפונקציות מאפשרות לשנות כל אלמנט תפריט לאחר שהצמדת את התפריט לחלון.

השינויים הנפוצים ביותר בתפריטים, הם בדרך כלל שינויי המחרוזות שמוצגות בתפריט מסוים, קביעה והסרה של תווי סימון לצידם של פריטי תפריט, נטרול פריטים בתפריט, או מחיקה והוספה של פריטים בתפריט. הפונקציה ModifyMenu מאפשרת

לבצע כמה מפעולות אלו על ידי קריאה לה. כתחליף, תוכל להשתמש בפונקציות יעודיות, כמו DeleteMenu או CheckMenuItem, כדי לערוך שינויים בפריטי התפריט. התקליטור שמצורף לספר מכיל את התוכנית **Delete_Item**, שמשתמשת בפקודות תפריט כדי לאפשר המשתמש להוסיף ולמחוק פריטים מתוך תפריט.

הפונקציה ModifyMenu

הפונקציה ModifyMenu מסוגלת לבצע שינויים שונים, בתפריט קיים. התוכניות יכולות להשתמש בפונקציה זו כדי להגדיר את התוכן, הצורה וההתנהגות של כל פריט בתפריט. משתמשים בפונקציה ModifyMenu בתוכניות כמו שרואים בהגדרה שלהלן:

```
BOOL ModifyMenu(
   HMENU hMenu,          // handle of menu
   UINT uPosition,       // menu item to modify
   UINT uFlags,          // menu item flag
   UINT uIDNewItem,      // menu item identifier or
                         // handle of drop-down menu
   LPCTSTR lpNewItem     // menu item content
);
```

הפרמטרים שמקבלת הפונקציה ModifyMenu מופיעים בטבלה 4.

טבלה 4: הפרמטרים שמקבלת הפונקציה **ModifyMenu**.

פרמטרים	תיאור
hMenu	מזהה את התפריט שרוצים לשנות.
uPosition	מגדיר את התפריט שרוצים לשנות לפי הערכים של הפרמטר uFlags.
uFlags	מגדיר דגלים ששולטים בפענוח הפרמטר uPosition והתכולה, ההופעה וההתנהגות של פריט תפריט. פרמטר זה חייב להיות שילוב של פרמטר אחד מאלה המפורטים בטבלה 5 ולפחות אחד מהפרמטרים שמפורטים בטבלה 6.
uIDNewItem	מגדיר אחד משני דברים, את המזהה של פריט התפריט ששונה, או אם נקבע הדגל MF_POPUP בפרמטר uFlags, הוא מגדיר את הידית של התפריט הנשלף למטה או של תפריט המשנה.
lpNewItem	מצביע לתוכן פריט התפריט ששונה. פענוח המובן של פרמטר זה תלוי בפרמטר uFlags אם הוא מכיל את הדגל MF_BITMAP, MF_OWNERDRAW או MF_STRING.

כפי שניתן לראות בטבלה 4, חייבים לקבוע ערך לפרמטר uFlags. הערך של uFlags חייב להיות שילוב של אחד הערכים שנמצאים בטבלה 5 וערך אחד או יותר מאלה שנמצאים בטבלה 6.

טבלה 5: הערכים האפשריים הנדרשים עבור הפרמטר **uFlags**.

ערך	פירוש
MF_BYCOMMAND	מציין שהפרמטר uPosition מעביר את המזהה של פריט התפריט. הדגל MF_BYCOMMAND הוא ברירת המחדל, אם לא הוגדר אף אחד מהדגלים MF_BYCOMMAND או MF_BYPOSITION.
MF_BYPOSITION	מציין שהפרמטר uPosition נותן את המקום היחסי מבוסס-אפס (Zero-Based Relative Position) של פריט התפריט.

בנוסף לערך המבוקש עבור הפרמטר uFlags, חובה להשתמש באופרטור OR הפועל על סיביות (Bitwise OR) כדי להציב ערך אחד או יותר מהערכים שנמצאים בטבלה 6.

טבלה 6: הערכים האפשריים של הפרמטר **uFlags**.

ערך	פירוש
MF_BITMAP	משתמש במפת סיביות כפריט בתפריט. הפרמטר lpNewItem מכיל את ידית מפת הסיביות.
MF_BYCOMMAND	מציין שהפרמטר uPosition מגדיר את מזהה פריט התפריט.
MF_BYPOSITION	מציין שהפרמטר uPosition מגדיר את המקום היחסי מבוסס-אפס (Zero-Based Relative Position) של הפריט החדש.
MF_CHECKED	מצמיד תו סימון לפריט. אם היישום מאפשר להשתמש במפות סיביות כתווי סימון (ראה את הפונקציה SetMenuItemNBitmaps), דגל זה מציג סימון של מפת סיביות ליד לפריט התפריט.
MF_DISABLED	מעביר פריט תפריט למצב לא פעיל, כך שהמשתמש לא יוכל לבחור בו ; אבל דגל זה אינו הופך את הפריט לאפור.
MF_ENABLED	מפעיל את פריט התפריט ומשחזר אותו ממצב אפור, כך שהמשתמש יוכל לבחור בו.
MF_GRAYED	מעביר פריט תפריט למצב לא פעיל, וגורם להצגתו באפור, כך שהמשתמש לא יוכל לבחור בו.

ערך	פירוש
MF_MENUBARBREAK	מתפקד כמו הדגל MF_MENUBREAK לשורת תפריט. עבור תפריטים הנשלפים למטה, תפריטי משנה, או תפריטי קיצור (Shortcut Menu), קו אנכי מפריד את העמודה החדש מהעמודה הקודמת.
MF_MENUBREAK	ממקם את הפריט בשורה חדשה (עבור שורות תפריט) או בעמוד חדש (עבור תפריטים הנשלפים למטה), תת-תפריט (Submenu, תפריט משנה) מבלי להפריד עמודות.
MF_OWNERDRAW	מגדיר שהפריט הוא פריט מסוג owner-drawn (פריט שהוא בן לפריט אחר, שמוצג כאשר האב שלו מצייר אותו). לפני שהתפריט מוצג בפעם הראשונה על ידי היישום, החלון שמכיל אותו מקבל את ההודעה WM_MEASUREITEM שמעבירה לו את רוחב וגובה פריט התפריט. אחר כך היישום שולח את ההודעה WM_DRAWITEM לפונקציית החלון של האב בכל פעם שהיישום חייב לעדכן את הופעת פריט התפריט.
MF_POPUP	מגדיר שפריט התפריט פותח תפריט נשלף למטה או תפריט משנה. הפרמטר uIDNewItem מגדיר את הידית של התפריט הנשלף למטה או של תפריט המשנה. משתמשים בדגל זה כדי להוסיף שם תפריט לשורת התפריט, או לפריט תפריט שפותח תפריט משנה של תפריט הנשלף למטה, תפריט משנה, או תפריט קיצור.
MF_SEPARATOR	מצייר קו מפריד אופקי. משתמשים בדגל זה רק כשמוסיפים פריטים לתפריט הנשלף למטה, תפריט משנה, או תפריט קיצור. אין באפשרות התוכניות להביא קו זה למצב לא פעיל, לצבוע אותו באפור, או להאיר אותו. Windows מתעלמת מן הפרמטרים lpNewItem ו-uIDNewItem.
MF_STRING	מצביע למחרוזת טקסט שבסופה NULL, כברירת מחדל.
MF_UNCHECKED	לא מציב תו סימון ליד הפריט (ברירת המחדל). אם היישום מספק תווי סימון של מפות סיביות (ראה את הפונקציה SetMenuItemBitmaps), דגל זה מציג מפת הסיבית של מצב לא מסומן (כלומר, הבחירה אינה מופעלת) ליד פריט התפריט.

כאשר הפונקציה ModifyMenu מחליפה פריט תפריט שפותח תפריט שנשלף למטה, או תפריט משנה, הפונקציה מפרקת את התפריט הקודם שנשלף למטה או את תפריט המשנה ומשחררת את הזיכרון שנתפס על ידי התפריט הקודם. בנוסף, היישום חייב לקרוא לפונקציה DrawMenuBar בכל פעם שתפריט משתנה, מבלי להתחשב אם הוא נמצא בחלון המוצג. כדי לשנות את המאפיינים של פריטי תפריט קיימים, יותר מהיר להשתמש בפונקציות CheckMenuItem ו-EnableMenuItem.

הערה: הפונקציה ModifyMenu אינה יכולה לקבל את קבוצות הדגלים הבאות באותה הקריאה.

MF_BYCOMMAND ו-MF_BYPOSITION
MF_DISABLED ,MF_ENABLED ו-MF_GRAYED
MF_BITMAP ,MF_STRING ,MF_OWNERDRAW ו-MF_SEPARATOR
MF_MENUBARBREAK ו-MF_MENUBREAK
MF_CHECKED ו-MF_UNCHECKED

כדי להבין יותר טוב את הטיפול של הפונקציה ModifyMenu, התבונן בתוכנית **Mod_Menu** שבתקליטור המצורף לספר זה. התוכנית **Mod_Menu.cpp** משנה את הפריט !Test לפריט התפריט !New Item כאשר המשתמש בוחר בו, אחר כך היא מטפלת בפריט התפריט !New Item אם המשתמש בוחר בו. תוכל לראות ששינוי זה מנוהל על ידי הפונקציה WndProc.

כפי שנראה בהמשך, התוכנית **Mod_Menu** משתמשת בפונקציה ModifyMenu, כדי לשנות את ערך המחרוזות של פריטי התפריט. בנוסף, הקוד בודק את מזהי התפריט, כדי לוודא שהפונקציה תפסה את הפריט החדש שנוצר.

שליטה בתפריטים באמצעות EnableMenuItem

התוכניות יכולות להשתמש בפונקציה ModifyMenu כדי לשלוט בצורה שבה הפריטים מוצגים בתפריט. עם זאת השימוש בפונקציות ייעודיות, עשוי לגרום לכך שהתוכנית תפעל מהר יותר. כדי לעזור לך לשנות פריט תפריט למצב פעיל (Enable), לנטרל הפעלה (Disable), או להפכו לאפור (Grayed), התוכניות יכולה להשתמש בפונקציה EnableMenuItem במקום הפונקציה ModifyMenu. משתמשים בפונקציה EnableMenuItem בתוכנית, כפי שמוצג בהגדרה שלהלן:

```
BOOL EnableMenuItem(
   HMENU hMenu,                // handle to menu
   UINT uIDEnableItem,         // menu item to enable,
                               // disable, or gray
   UINT uEnable,               // menu item flags
);
```

טבלה 7: הערכים האפשריים שמקבל הפרמטר **uEnable**.

דגל	תיאור
MF_BYCOMMAND	מציין שהפרמטר uIDEnableItem מעביר את מזהה פריט התפריט. אם המשתמש אינו מגדיר את הדגל MF_BYCOMMAND או את הדגל MF_BYPOSITION. במקרה זה הדגל MF_BYCOMMAND הוא ברירת המחדל.
MF_BYPOSITION	מציין שהפרמטר uIDEnableItem מעביר את המקום היחסי מבוסס-אפס (Zero-Based Relative Position) של פריט התפריט.
MF_DISABLED	מציין שפריט התפריט במצב לא פעיל, אבל לא במצב אפור, ולכן המשתמש לא יכול לבחור בו.
MF_ENABLED	מציין ש-Windows צריכה להפוך את הפריט למצב פעיל ולשחזר אותו ממצב אפור, כך שמשתמש יכול לבחור בו.
MF_GRAYED	מציין שפריט התפריט במצב לא פעיל, ולכן המשתמש אינו יכול לבחור בו.

הפרמטר hMenu מכיל את ידית התפריט שרוצים לשנות. הפרמטר uIDEnableItem מגדיר את פריט התפריט שברצונך **לאפשר הפעלתו** (Enable), **לנטרל את הפעלתו** (Disable), או להביאו למצב **אפור** (Grayed) לפי הערך בפרמטר uEnable. הפרמטר uIDEnableItem מגדיר **פריט בשורת התפריט** (Item In A Menu Bar), **תפריט** (Menu), או **תפריט משנה** (Submenu). הפרמטר uEnable מגדיר דגלים ששולטים בפענות ערכו של הפרמטר uIDEnableItem ומציין מתי פריט התפריט במצב פעיל, במצב לא פעיל, או במצב אפור. הפרמטר uEnable חייב להיות אחד מהשילובים האלה: MF_BYCOMMAND או MF_BYPOSITION עםMF_ENABLED , MF_DISABLED או MF_GRAYED, כמו שרואים בטבלה 7.

היישום חייב להשתמש בדגל MF_BYCOMMAND כדי להגדיר את ידית התפריט הנכונה. אם התוכנית הגדירה את **ידית התפריט** (Menu Handle) של **שורת התפריט** (Menu Bar), Windows מבצעת את הפעולה כנגד **הפריט שנמצא בשורת התפריט הראשית** (Top-Level Menu Item). כדי לקבוע מצב של פריט בתפריט שנשלף למטה או בתפריט משנה, חובה על היישום להגדיר את ידית התפריט שנשלף למטה, או את ידית תפריט המשנה.

כאשר היישום מגדיר את הדגל MF_BYPOSITION, Windows בודקת את כל הפריטים שפותחים תפריט משנה מתוך התפריט. לכן, אם אין שמות כפולים של פריטי תפריט, אפשר להסתפק בהגדרה של ידית שורת התפריט עבור שורת התפריט.
כדי להבין טוב יותר את הטיפול ש-EnableMenuItem מבצעת, התבונן בתוכנית **Enab_Dis** שבתקליטור המצורף לספר. תוכנית זו משתמשת בפונקציה EnableMenuItem כדי להביא את הפריט IDM_ITEM1 למצב פעיל או למצב לא פעיל.

כרגיל, הפונקציה WndProc שבקובץ **Enab_Dis.cpp** מכילה את הקוד שמבצע פעולות אלו, כפי שמוצג להלן:

```
case IDM_TEST :
  {
    HMENU hMenu  = GetMenu(hWnd);
    UINT  uState = GetMenuState(hMenu, IDM_ITEM1,
                                MF_BYCOMMAND);
    if (uState & MFS_GRAYED)
      EnableMenuItem(hMenu, IDM_ITEM1,
                     MFS_ENABLED | MF_BYCOMMAND);
    else
      EnableMenuItem(hMenu, IDM_ITEM1,
                     MFS_GRAYED | MF_BYCOMMAND);
  }
break;
```

הרחבת תפריט באמצעות AppendMenu

התוכניות יכולות לבצע מספר רב של פעולות שונות על תפריטים גם לאחר שהתוכנית קשרה והציגה את התפריט בחלון. אחת הפעולות הנפוצות ביותר שהתוכניות מבצעות על תפריטים זו הוספת פריטים לתפריט. הפונקציה AppendMenu מוסיפה לסוף של שורת התפריט פריט חדש שמוגדר על ידך. באפשרותך להגדיר תפריט הנשלף למטה, תפריט משנה, או תפריט קיצור. תוכל גם להשתמש בפונקציה AppendMenu כדי להגדיר את התוכן, ההופעה וההתנהגות של פריט התפריט, כמו שרואים להלן:

```
BOOL AppendMenu (
   HMENU hMenu,       // handle to menu to be changed
   UINT uFlags,       // menu-item flag
   UINT uIDNewItem,   // menu-item identifier or handle
                      // of drop-down menu or submenu
   LPCTSTR lpNewItem  // menu-item content
);
```

בדרך כלל הפרמטר hMenu מציין שורת תפריט, תפריט הנשלף למטה, תפריט משנה, או תפריט קיצור ש-AppendMenu עומדת לשנות. הפרמטר uFlags מגדיר דגלים לשליטה בהופעה והתנהגות של פריט התפריט החדש. הפרמטר uIDNewItem מגדיר אחד משני דברים: את המזהה של פריט התפריט החדש, או אם נקבע הערך MF_POPUP לפרמטר uFlags הוא מגדיר את ידית התפריט הנשלף למטה או תפריט המשנה. הפרמטר lpNewItem מגדיר את תוכן פריט התפריט החדש. הצורה שבה הפונקציה AppendMenu מפרשת את הפרמטר lpNewItem תלויה בדגל שמכיל הפרמטר uFlags: הדגל MF_BITMAP, MF_OWNERDRAW או MF_STRING. עיין בטבלה 6.

היישום חייב לקרוא לפונקציה DrawMenuBar בכל פעם שהתפריט משתנה, ללא קשר אם הוא נמצא בחלון שמוצג כרגע. באפשרות התוכנית לקבוע מספר דגלים לפרמטר uFlags כפי שמפורט בטבלה 8.

✍ **הערה:** הפונקציה AppendMenu אינה מאפשרת להשתמש בשילוב הדגלים כפי שמוצג להלן:

```
MF_BYCOMMAND-ו MF_BYPOSITION
MF_DISABLED ,MF_ENABLED-ו MF_GRAYED
MF_BITMAP ,MF_STRING ,MF_OWNERDRAW-ו MF_SEPARATOR
MF_MENUBARBREAK-ו MF_MENUBREAK
MF_CHECKED-ו MF_UNCHECKED
```

טבלה 8: הערכים האפשריים של הפרמטר **uFlags**.

ערך	פירוש
MF_BITMAP	משתמש במפת סיביות כפריט בתפריט. הפרמטר lpNewItem מכיל את הידית של מפת הסיביות.
MF_CHECKED	מצמיד תו סימון לפריט. אם היישום מאפשר להשתמש בתווי סימון של מפות סיביות, דגל זה מציג סימון של מפת סיביות צמוד לפריט התפריט.
MF_DISABLED	מעביר פריט תפריט למצב לא פעיל, כך שהמשתמש לא יוכל לבחור בו, אבל דגל זה לא הופך את הפריט לאפור.
MF_ENABLED	מפעיל את פריט התפריט ומשחזר אותו ממצב אפור, כדי שהמשתמש יוכל לבחור בו.
MF_GRAYED	מעביר פריט תפריט למצב לא פעיל וגורם להצגתו באפור, כדי שהמשתמש לא יוכל לבחור בו.
MF_MENUBARBREAK	מתפקד כמו הדגל MF_MENUBREAK עבור שורת תפריט. עבור תפריטים הנשלפים למטה, תפריט משנה, או תפריטי קיצור (Shortcut Menu), קו אנכי מפריד בין העמודה החדשה לבין העמודה הקודמת.
MF_MENUBREAK	מציבה את הפריט בשורה חדשה (עבור שורות תפריט) או בעמוד חדש (עבור תפריטים הנשלפים למטה, **תת-תפריט** (Submenu), או תפריט משנה, או תפריטי קיצור), מבלי להפריד עמודות.

ערך	פירוש
MF_OWNERDRAW	מגדיר שהפריט הוא פריט owner-drawn (פריט שהוא בן לפריט אחר שמוצג כאשר האב שלו מצייר אותו). לפני שהתפריט מוצג בפעם הראשונה על ידי היישום, החלון שמכיל את התפריט מקבל את ההודעה WM_MEASUREITEM שמעבירה לו את רוחב וגובה פריט התפריט. לאחר מכן היישום שולח ההודעה WM_DRAWITEM לפונקציית החלון של האב בכל פעם שהיישום חייב לעדכן את ההופעה של פריט התפריט.
MF_POPUP	מגדיר שפריט התפריט פותח תפריט נשלף למטה או תפריט משנה. הפרמטר uIDNewItem מגדיר את ידית התפריט הנשלף למטה או של תפריט המשנה. משתמשים בדגל זה כדי להוסיף שם תפריט לשורת התפריט או לפריט תפריט שפותח תפריט משנה של תפריט הנשלף למטה, תפריט משנה, או תפריט קיצור.
MF_SEPARATOR	מצייר קו מפריד אופקי. משתמשים בדגל זה רק בתפריט הנשלף למטה, תפריט משנה, או תפריט קיצור. אין באפשרות התוכניות להביא קו זה למצב לא פעיל, לצבוע אותו באפור, או לבחור בו. הפונקציה AppendMenu מתעלמת מן הפרמטרים lpNewItem ו-uIDNewItem כאשר מגדירים את הסוג MF_SEPARATOR.
MF_STRING	מגדיר שפריט התפריט הוא מחרוזת טקסט; הפרמטר lpNewItem מצביע אל המחרוזת.
MF_UNCHECKED	לא שם תו סימון ליד הפריט (ברירת המחדל). אם היישום מספק תווי סימון של מפות סיביות (ראה הפונקציה SetMenuItemBitmaps), דגל זה מציג את מפת הסיביות שמסמנת מצב לא מסומן (כלומר הבחירה לא מופעלת) ליד פריט התפריט.

כדי להבין יותר טוב את פעולת הפונקציה AppendMenu, התבונן בתוכנית **Add_New** (הנמצאת בתקליטור). היא מוסיפה את פריט התפריט New Item בשורה נפרדת, בכל פעם שהמשתמש בוחר בפריט התפריט !Test. הפונקציה WndProc לוכדת את הבחירה בפריט !Test ומשתמשת ב-AppendMenu כדי להוסיף פריט חדש, כפי שמוצג להלן:

```
case IDM_TEST :
   // Add new menu option on a new line.
   AppendMenu( GetMenu( hWnd ),
               MF_STRING | MF_MENUBARBREAK,
               120, "New Item" );
   DrawMenuBar( hWnd );
break;
```

מחיקת פריטי תפריט שנבחרו, עם הפונקציה DeleteMenu

בסעיף הקודם השתמשת בפונקציה AppendMenu כדי להוסיף פריטים לתפריט מסוים. אפשר להשתמש גם בפונקציה DeleteMenu כדי למחוק פריט מתפריט. אם פריט התפריט פותח תפריט הנשלף למטה או תפריט משנה, הפונקציה DeleteMenu מפרקת את הידית לתפריט או לתפריט המשנה, ומשחררת את הזיכרון שהם השתמשו בו.

משתמשים בפונקציה DeleteMenu כמו בהגדרה שלהלן:

```
BOOL DeleteMenu (
   HMENU hMenu,        // handle to menu
   UINT uPosition,     // menu item identifier or position
   UINT uFlags         // menu item flag
);
```

כמו תמיד, היישום חייב לקרוא לפונקציה DrawMenuBar בכל פעם שהתפריט משתנה, ללא תלות אם התפריט נמצא בחלון שמוצג כרגע. כדי להבין טוב יותר את הטיפול ש-DeleteMenu מבצעת, התבונן בתוכנית **Add_Del**, שבתקליטור המצורף לספר זה, אשר מוסיפה שלושה פריטים חדשים לתפריט בזמן יצירת החלון ואחר כך מוחקת את הפריטים החדשים האלה כאשר המשתמש בוחר פריט. התוכנית מבצעת את טיפול זה במסגרת ההודעות WM_CREATE ו-WM_COMMAND שבהוראות ה-switch שבפונקציה WndProc, כפי שמוצג להלן:

```
case WM_CREATE :
  {
     HMENU hMenu = GetMenu( hWnd );

     AppendMenu( hMenu, MFT_STRING, IDM_ITEM1, "Item&1" );
     AppendMenu( hMenu, MFT_STRING, IDM_ITEM2, "Item&2" );
     AppendMenu( hMenu, MFT_STRING, IDM_ITEM3, "Item&3" );
  }
  break;

  case WM_COMMAND :
     switch( LOWORD( wParam ) )
       {
          case IDM_ITEM1 :
          case IDM_ITEM2 :
          case IDM_ITEM3 :
```

```
        HMENU hMenu = GetMenu( hWnd );
        DeleteMenu( hMenu, LOWORD(wParam), MF_BYCOMMAND );
        DrawMenuBar( hWnd );
        break;
        ...
```

העמקה - מבנה קבצי משאבים

למעשה, כל תוכנית Windows משתמשת במשאבים. Windows שומרת את המידע אודות המשאבים בקבצים, עוזרת לך באחזקת הקבצים בצורה מאורגנת ונגישה ושומרת שהקוד שלך לא יהיה מסורבל. בדרך כלל מאוחסן המידע אודות המשאבים שהתוכנית משתמשת בהם ב**קבצי משאבים - rc** (Resource Files). השימוש בקבצי המשאבים יעיל למדי, כי לרוב התוכנית טוענת את מידע המשאב לזיכרון רק כאשר היא זקוקה למשאב במהלך הביצוע.

בדרך כלל, מהדר המשאבים (בדרך כלל הוא נקרא rc.exe) מהדר את קובץ המשאבים והופך אותו לקובץ RES. **המקשר** (Linker) מקשר את קובץ RES לסופו של קובץ ההפעלה של התוכנית שעבר הידור בהצלחה, ואז כל המשאבים שהוגדרו בקובץ המשאבים הופכים להיות זמינים לשימוש התוכנית בזמן הפעלתה.

קובץ המשאבים יכול להכיל חמישה סוגי משאבים המוגדרים **בשפת תסריט** (Script) ונכתבים כל אחד בשורה אחת: MESSAGETABLE-ו FONT ,ICON ,CURSOR ,BITMAP. כל אחת מהוראות אלו טוענת אל טבלת המשאבים שבזיכרון את הקובץ שנושא את הסוג המוגדר בהוראה. כאשר משאב נכלל בקובץ המשאבים, התוכנית יכולה להשתמש בפונקציות טעינה כמו LoadIcon, כדי לגשת אליו ולהשתמש בו. בדרך כלל סוג של משאב המוגדר בשורה אחת נראה כך:

```
MYAPP      ICON     DISCARDABLE       "GENERIC.ICO"
```

בנוסף לחמשת סוגי המשאבים שמוגדרים **בשפת תסריט** באמצעות שורה אחת, יש חמישה סוגי הגדרות בשפת תסריט מרובות שורות (Multiple-Line): ACCELERATOR, DIALOG, MENU, RCDATA ו-STRINGTABLE. ששת הסעיפים הבאים מסבירים איך להשתמש בסוגים STRINGTABLE ו-RCDATA.

קל לזהות את סוגי המשאבים שמוגדרים בקבצי המשאבים באמצעות שורות רבות. כל סוג של משאב שמוגדר על ידי שורות רבות מכיל **הוראת סוג** (Type Statement), בלוק מסוג BEGIN-END, והוראות בתוך הבלוק BEGIN-END או בלוקים נוספים מסוג BEGIN-END, כמו שרואים בדוגמה הזו:

```
NEWMENU MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",     IDM_EXIT
    END
    MENUITEM "&Old Menu!",    IDM_OLD
END
```

מבוא לטבלאות מחרוזות

למדת שאחד מסוגי המשאבים מרובה השורות שקבצי המשאבים תומכים בו הוא STRINGTABLE. רוב היישומים משתמשים בסדרה של מחרוזות תווים בהודעות ובפלט הכולל תווים. Windows מספקת **טבלאות מחרוזות** (String Tables) כתחליף לשיטה המקובלת של מיקום מחרוזות באזור הנתונים הסטטיים של התוכנית. לכן, באפשרות התוכנית להגדיר מחרוזות תווים בקובץ המשאבים, ולתת למחרוזות **ערך מזהה** (ID Value) כפי שמוצג להלן:

```
STRINGTABLE
BEGIN
    IDS_STRINGBASE1    "Simple String Sample."
    IDS_STRINGBASE2    "Jamsa's C/C++ Programmer's Bible"
    IDS_STRINGBASE3    "Jamsa and Klander"
END
```

בנוסף להגדרות התוכן שלהן במשאב, מגדירים בדרך כלל את הערכים המזהים של המחרוזות (כמו IDS_STRING3) בקובץ כותר נפרד, שמכלילים אותו בקובץ המשאבים והמודול (או המודולים) שעושים שימוש במחרוזות אלו. כשהיישום חייב לגשת לנתונים, צריך להשתמש בפונקציה LoadString כדי להעתיק את נתוני התווים מקובץ המשאבים למאגר בזיכרון. מחרוזות ב**טבלת מחרוזות** (String Table) יכולות להכיל תווי בקרה (כמו למשל, טאבים ושורה חדשה), וגם תווי בקרה להדפסה.

יש מספר יתרונות תכנות עיקריים לשימוש בטבלאות מחרוזות. היתרון העיקרי הוא בהקטנת דרישות הזיכרון עבור התוכנית. מכיון שהתוכנית אינה טוענת את המחרוזות עד שהיא צריכה אותן, אין צורך לאחסן את המחרוזות באזור הנתונים הסטטי שלה. לכן, צריך להימנע מלהעתיק את הנתונים של טבלת המחרוזות ל**מאגר זיכרון סטטי** (Static Memory Buffer), כי אם עושים זאת, מבטלים למעשה את יתרון השימוש בטבלאות מחרוזות. מה שכן צריך לעשות, הוא להעתיק את נתוני טבלת המחרוזות למשתנה מקומי (משתנה מחסנית), או לזיכרון הגלובלי.

יתרון עיקרי נוסף לשימוש בטבלאות מחרוזת הוא התמיכה שלהן במספר שפות. Windows API תומכת ב**משאבים מרובי שפות** (Multi-Lingual Resources) עם יישום יחיד. כלומר, ניתן להפיץ את אותו קובץ הפעלה של התוכנית במספר מדינות (הדוברות בשפות שונות), מבלי שתצטרך לשנות אותו. בסעיפים הבאים נשתמש בטבלאות מחרוזות כדי להחזיק את מידע החלון.

משאבים מותאמים אישית (Custom Resources)

אחד מסוגי המשאבים מרובי השורות הוא RCDATA. אפשר להשתמש בסוג זה כדי לאחסן סוגים אחרים של נתונים סטטיים, ובמיוחד נתונים בינריים גולמיים. לדוגמה, קטע הקוד הבא מאחסן מספר של סוגי נתונים שונים באמצעות המשאב DataID:

```
DataID RCDATA
  BEGIN
     3
     40
     0x8232
     "string data (continued)..."
     "More String Data\0"
  END
```

באפשרותך להכיל משאב נתונים מותאם אישית בקובץ התוכנית, או לקרוא אותו מקובץ נתונים חיצוני, אך המקום הטוב ביותר לאחסנת משאב נתונים מותאם אישית הוא בקובץ חיצוני. מהדר המשאבים יכול במקרה זה להוסיף את התוכן של הקבצים החיצוניים למשאב המידע כאשר הוא מהדר את קובץ המשאבים. לדוגמה, שתי ההוראות הבאות מגדירות את סוגי המשאבים TEXT ו-METAFILE המותאמים אישית, קובעות מזהה לסוגים, ואז מוסיפות את המשאבים לקובץ המשאבים:

```
happy               TEXT               "happydog.txt"
picture             METAFILE           "happypic.wmf"
```

כאשר מגדירים סוגי משאבים מותאמים אישית, משתמשים בפונקציה FindResource יחד עם הפונקציה LoadResource כדי לטעון משאבים מותאמים אישית לתוכניות.

טעינת טבלאות משאבים לתוכניות באמצעות LoadString

למדת שאפשר ליצור טבלאות מחרוזות עם קבצי המשאבים. עליך להציב לכל מחרוזת בטבלת המחרוזות ערך של מזהה מסוים, שבדרך כלל מגדירים אותו בקובץ הכותר של התוכנית, כפי שמוצג להלן:

```
#define IDM_EXIT      100
#define IDM_TEST      200
#define IDM_ABOUT     301
```

לאחר שמגדירים טבלת מחרוזות, משתמשים בפונקציה LoadString כדי לטעון משאב מחרוזות מקובץ ההפעלה שקשור למודול שהוגדר, להעתיק את המחרוזות לתוך מאגר, ולהוסיף את NULL כתו מסיים למאגר. משתמשים בפונקציה LoadString, כמו שמוצג בהגדרה שלהלן:

```
int LoadString(
   HINSTANCE hInstance  // handle of module containing
                        // string resource
   UINT uID             // resource identifier
   LPTSTR lpBuffer      // address of buffer for resource
   int nBufferMax       // size of buffer
);
```

הפרמטר hInstance מזהה את מופע המודול שקובץ ההפעלה שלו מכיל את משאב המחרוזות. הפרמטר uID הוא מספר שלם שמזהה את המחרוזות שהפונקציה LoadString צריכה לטעון. הפרמטר lpBuffer מצביע למאגר שמקבל את המחרוזות. לבסוף, הפרמטר nBufferMax מגדיר את גודל המאגר בבתים (גרסת ANSI), או בתווים (גרסת Unicode). הפונקציה מקצצת ומוסיפה NULL מסיים למחרוזות במקרה שהמחרוזות יותר ארוכה ממספר התווים שציינת.

אם LoadString מצליחה, היא מחזירה את מספר הבתים (גרסת ANSI) או את מספר התווים (גרסת Unicode) שהעתיקה למאגר, לא כולל את התו המסיים NULL; או אפס, אם משאב המחרוזות לא קיים. כדי להשיג מידע נוסף אודות השגיאה, אפשר לקרוא לפונקציה GetLastError. כדי להבין טוב יותר את הטיפול שהפונקציה LoadString מבצעת, התבונן בקטע הקוד הבא מתוכנית **LoadStrg** שנמצאת בתקליטור המצורף לספר זה (בתיקיה Books\59285\Nis-A):

```
case IDM_TEST :
        {
          char  szString[40];
          SHORT idx;
          for (idx = IDS_STRINGBASE;
               idx < IDS_STRINGBASE+3; idx++)
         {
            LoadString(hInst, idx, szString, 40 );
            MessageBox(hWnd, szString, "String Loaded",
                        MB_OK | MB_ICONINFORMATION );
         }
      }
      break;
```

התוכנית **LoadStrg** תטען ותציג שלוש מחרוזות נפרדות כאשר המשתמש בוחר באפשרות !Test. התוכנית תציג כל מחרוזות בתיבת הודעה נפרדת.

הצגת תוכן קבצי המשאבים

אפשר לאחסן בקובץ המשאבים מספר רב של סוגי נתונים מותאמים אישית. הפונקציה EnumResourceNames מחפשת כל סוג של משאב שהוגדר במודול לפי הערך שהצבת לפרמטר lpszType. אחר כך היא מוסרת לפונקציית משוב (Callback Function) המוגדרת על ידי היישום, את השם של כל משאב שהיא מאתרת. הפונקציה EnumResourceNames ממשיכה לסקור את שמות המשאבים, עד שפונקציית המשוב מחזירה False, או עד שהיא מסיימת לסקור את כל שמות המשאבים.

את הפונקציה EnumResourceNames כותבים כמו בדוגמה שלהלן:

```
BOOL EnumResourceNames(
   HINSTANCE hModule, // resource-module handling
   LPCTSTR lpszType   // pointer to resource type
   ENUMRESNAMEPROC lpEnumFunc, // pointer to callback
                   // funcion
   LONG lParam        // application-defined parameter
);
```

הפונקציה EnumResourceNames מקבלת את הפרמטרים שמפורטים בטבלה 9.

טבלה 9: הפרמטרים שהפונקציה **EnumResourceName**s מקבלת.

פרמטר	תיאור
hModule	מזהה את המודול שקובץ ההפעלה שלו מכיל את המשאבים שהפונקציה EnumResourceNames צריכה לסקור את השמות. אם פרמטר זה הוא NULL, הפונקציה מונה את שמות המשאב במודול שהשתמשנו בו, כדי ליצור את התהליך הנוכחי.
lpszType	מצביע למחרוזת המסתיימת ב-NULL, אשר מגדירה את סוג המשאב שעבורו הפונקציה EnumResourceNames סוקרת את השמות. עבור סוגי משאב סטנדרטים, פרמטר זה יכול לקבל את אחד הערכים שמפורטים בטבלה 10.
lpEnumFunc	מצביע לפונקציית המשוב שנקראת על ידי הפונקציה EnumResourceNames עבור כל שם משאב שסורקים.
lParam	מציין ערך שמוגדר על ידי היישום שהפונקציה EnumResourceNames מוסרת לפונקציית המשוב. באפשרות התוכניות להשתמש בפרמטר זה בזמן שהן בודקות שגיאות.

כמו שרואים בטבלה 9, הפרמטר lpszType יכול לקבל ערך אחד מתוך מספר ערכים שמפורטים בטבלה 10.

טבלה 10: הערכים האפשריים שהפרמטר **lpszType** יכול לקבל.

ערך	פירוש
RT_ACCELERATOR	טבלת מקשים מהירים (Accelerator Table).
RT_ANICURSOR	סמן הנפשה (Animated Cursor).
RT_ANICON	סמל הנפשה (Animated Icon).
RT_BITMAP	משאב מפת סיביות (Bitmap Resource).
RT_CURSOR	משאב סמן שתלוי בחומרה (HardwAre-Dependent Cursor Resource).
RT_DIALOG	תיבת דו-שיח (Dialog Box).
RT_FONT	משאב גופן (Font Resource).
RT_FONTDIR	משאב גופן לתיקיה (Font Directory Resource).
RT_GROUP_CURSOR	משאב סמן שאינו תלוי בחומרה (Hardware-Independent Cursor Resource).
RT_GROUP_ICON	משאב סמל שאינו תלוי בחומרה (Hardware-Independent Cursor Icon).
RT_ICON	משאב סמל תלוי בחומרה (Hardware-Dependent Icon Resource).
RT_MENU	משאב תפריט (Menu Resource).
RT_MESSAGETABLE	כניסה בטבלת הודעות (Message-Table Entry).
RT_PLUGPLAY	משאב הכנס-הפעל (Plug and Play Resource).
RT_RCDATA	משאב שמוגדר על ידי היישום (נתונים גולמיים, Raw Data).
RT_STRING	כניסה של טבלת מחרוזת (String-Table Entry).
RT_VERSION	משאב גירסה (Version Resource).
RT_VXD	מנהל התקן וירטואלי (VDX, VIRTUAL Device Driver).

כדי להבין יותר טוב איך פועלת הפונקציה EnumResourceNames, התבונן בתוכנית **3_pics** שבתקליטור המצורף לספר זה. קובץ המשאבים של התוכנית מכיל שלוש מפות סיביות. התוכנית משתמשת בפונקציה EnumResourceNames ובפונקציית המשוב PaintBitmaps כדי למנות את המשאבים מסוג RT_BITMAP שבקובץ המשאבים **3_pics.rc**.

הקוד הבא מציג את פונקציית המשוב:

```
BOOL CALLBACK PaintBitmaps(HANDLE hModule, LPCTSTR lpszType,
                           LPCTSTR lpszName, LONG lParam)
{
   HBITMAP hBitmap = LoadBitmap( hModule, lpszName );
   if ( hBitmap )
   {
      BITMAP bm;
      HDC    hMemDC;
      HWND   hWnd = (HWND)lParam;
      HDC    hDC  = GetDC( hWnd );
      HFONT  hOldFont;

      // Get the size of the bitmap.
      GetObject( hBitmap, sizeof( BITMAP ), &bm );

      // Create a memory DC to select the bitmap into.
      hMemDC = CreateCompatibleDC( hDC );
      SelectObject( hMemDC, hBitmap );

      // Display the bitmap, stretching it to 50X50 pixels.
      StretchBlt( hDC, gnPos, 0, 50, 50, hMemDC, 0, 0,
                  bm.bmWidth, bm.bmHeight, SRCCOPY );

      // Display the bitmap name.
      hOldFont = SelectObject(hDC,
                        GetStockObject(ANSI_VAR_FONT));
      TextOut(hDC, gnPos, 60, lpszName, strlen(lpszName));
      SelectObject( hDC, hOldFont );
      DeleteDC( hMemDC );
      ReleaseDC( hWnd, hDC );
      DeleteObject( hBitmap );
      gnPos += 100;
   }
  return( TRUE );
}
```

פונקציית המשוב PaintBitmaps מבצעת עיבוד חשוב. היא משרטטת במסך כל מפת סיביות שהפונקציה EnumResourceNames סוקרת.

שימוש ב-EnumResourceTypes עם קבצי המשאבים

למדת על הפונקציה EnumResourceNames, שאפשר להשתמש בה בתוכניות כדי לסקור בקובץ המשאבים את המשאבים השונים מסוג נתון. אך ייתכנו מקרים בהם התוכנית לא תדע מראש את סוגי המשאבים השונים שנמצאים בקובץ המשאבים. במקרים כאלה, באפשרות התוכניות להשתמש בפונקציה EnumResourceTypes, כדי לחפש מודול הכולל את המשאבים הדרושים. היא מעבירה כל סוג משאב שהיא מוצאת אל פונקציית משוב שמוגדרת על ידי היישום. הפונקציה EnumResourceTypes ממשיכה לסקור את סוגי המשאבים עד שפונקציית המשוב מחזירה False, או עד שהיא מסיימת לסקור את כל סוגי המשאבים. משתמשים בפונקציה EnumResourceTypes בתוכניות, כפי שמוצג להלן:

```
BOOL EnumResourceTypes(
   HMODULE hModule,             // resource-module handle
   ENUMRESTYPEPROC lpEnumFunc, // pointer to callback
                               // funcion
   LONG lParam                 // application-defined parameter
);
```

כשם שהפונקציה EnumResourceNames משתמשת בפונקציית משוב, כך גם הפונקציה EnumResourceTypes משתמשת בפונקציית משוב. הגדרת פונקציית המשוב שבה משתמשים עם הפונקציה EnumResourceTypes מוצגת בדוגמה הבאה (תוכל כמובן, לבחור שם אחר לפונקציית המשוב שלך, במקום EnumResTypeProc):

```
BOOL CALLBACK EnumResTypeProc(
   HANDLE hModule,    // resource-module handle
   LPTSTR lpszType,   // pointer to resource type
   LONG lParam        // application-defined parameter
);
```

הפרמטר lpszType מצביע למחרוזת המסתיימת ב-NULL שמגדירה את שם הסוג של המשאב שעבורו הפונקציה מונה את הסוג. עבור סוגים של משאבים סטנדרטיים, הפרמטר lpszType יכול להכיל את אחד הערכים המפורטים בטבלה 9.

כדי להבין יותר טוב את מהלך העיבוד של הפונקציה EnumResourceTypes, התבונן בתוכנית **EnumRest** שבתקליטור המצורף לספר זה. תוכנית זו רושמת ב**תיבת רשימה** (List Box) את כל הסוגים השונים שהיא מוצאת בקובץ משאבים נתון. קטע הקוד שלהלן מציג את פונקציית המשוב ListResourceTypes, שהפונקציה EnumResourceTypes משתמשת בה כפונקציית משוב:

```
BOOL CALLBACK ListResourceTypes(HANDLE hModule, LPTSTR
lpszType, LONG lParam )
{
  LPTSTR lpAddString = lpszType;
  HWND   hListBox    = (HWND)lParam;
```

```
    // Check to see if the resource type is a pre-defined
    // type. If it is set lpAddString to a descriptive string.
  switch(LOWORD(lpszType) )
  {
    case RT_ACCELERATOR : lpAddString = "Accelerator"; break;
    case RT_BITMAP      : lpAddString = "Bitmap"; break;
    case RT_DIALOG      : lpAddString = "Dialog"; break;
    case RT_FONT        : lpAddString = "Font"; break;
    case RT_FONTDIR     : lpAddString = "FontDir"; break;
    case RT_MENU        : lpAddString = "Menu"; break;
    case RT_RCDATA     : lpAddString = "RC Data"; break;
    case RT_STRING     : lpAddString = "String Table"; break;
    case RT_MESSAGETABLE : lpAddString = "Message Table";
                       break;
    case RT_CURSOR     : lpAddString = "Cursor"; break;
    case RT_GROUP_CURSOR : lpAddString = "Group Cursor";
                       break;
    case RT_ICON       : lpAddString = "Icon"; break;
    case RT_GROUP_ICON : lpAddString = "Group Icon"; break;
    case RT_VERSION    : lpAddString = "Version Information";
                       break;
    }
    SendMessage ( hListBox, LB_INSERTSTRING, (WPARAM)-1,
                       (LPARAM)lpAddString );
    return ( TRUE );
}
```

פונקציית המשוב מקבלת את הערך lpszType מהפונקציה EnumResourceTypes וממירה אותו לערך של מחרוזת המוכר יותר למשתמש, כמו "Icon" או "Menu".

טעינת משאבים לתוכניות באמצעות FindResource

התוכניות יכולות להגדיר בקובץ המשאבים מספר כלשהו של משאבים מותאמים אישית. בסעיפים קודמים השתמשנו בפונקציות EnumResourceNames ו-EnumResourceTypes כדי לקבל את רשימת כל השמות של משאב מסוג נתון, ואת רשימת כל סוגי המשאבים שבקובץ המשאבים. כתחליף, יכולות התוכניות להשתמש בפונקציות FindResource ו-LoadResource, כדי להשיג את אותה תוצאה. הפונקציה FindResource קובעת את מקום המשאב על פי הסוג והשם שמוגדרים במודול שציינת.

את הפונקציה FindResource כותבים כפי שמוצג להלן:

```
HRSRC FindResource(
  HMODULE hModule,       // resource-module handle
  LPCTSTR lpName,        // pointer to resource name
  LPCTSTR lpType         // pointer to resource type
);
```

כאשר הפונקציה FindResource מצליחה, הערך המוחזר הוא ידית לבלוק המידע של המשאב הרצוי. כדי לקבל את ידית המשאב, צריך למסור לפונקציה LoadResource את הידית שהפונקציה FindResource מחזירה. אם הפונקציה נכשלת, הערך המוחזר הוא NULL.

אם מילת הסדר-הגבוה של הפרמטר lpName או של הפרמטר lpType שווה לאפס, מילת הסדר-הנמוך מגדירה מספר שלם, שהוא המזהה של השם או של סוג המשאב הנתון; אחרת, פרמטרים אלה יהיו מצביעים ארוכים למחרוזות המסתיימות ב-NULL. אם התו הראשון של המחרוזת הוא התו #, שאר התווים מייצגים מספר עשרוני שמגדיר את המספר השלם אשר מזהה את שם המשאב, או את סוגו. לדוגמה, המחרוזת "‎#258" מייצגת את המזהה, שהוא המספר השלם 258.

היישומים צריכים לצמצם את צריכת הזיכרון של המשאבים על ידי ההתייחסות למשאבים עם מזהים של מספרים שלמים, ולא לפי שם. יישום יכול להשתמש בפונקציה FindResource כדי למצוא כל סוג של משאב, אך צריך להשתמש ב-FindResource רק אם על היישום לגשת למשאב מידע בינרי כשהוא מבצע קריאות עוקבות לפונקציות LoadLibrary ו-LockResource. כדי ללמוד יותר על LoadLibrary ו-LockResource, התבונן בתיעוד הנלווה למהדר שלך בעזרה המקוונת.

כדי להשתמש במשאב מייד, על היישום להשתמש באחת מפונקציות המשאב המיוחדות (Resource-Specific) המפורטות בטבלה 11, כדי למצוא ולטעון את המשאבים בקריאה אחת.

טבלה 11: פונקציות **Load** המיוחדות של המשאב (resource-specific).

פונקציה	פעולה
FormatMessage	טעינה ועיצוב כניסה בטבלת הודעות (A Message Table Entry).
LoadAccelerators	טעינה של טבלת מקשים מהירים.
LoadBitmap	טעינת משאב מפת סיביות.
LoadCursor	טעינת משאב סמן.
LoadIcon	טעינת משאב סמל.
LoadMenu	טעינת משאב תפריט.
LoadString	טעינת טבלת-מחרוזות (String-Table).

לדוגמה, יישום יכול להשתמש בפונקציה LoadIcon כדי לטעון סמל שיוצג על המסך. אך, היישום צריך להשתמש ב-FindResource ו-LoadResource אם היישום טוען את הסמל כדי להעתיק את הנתונים שלו ליישום אחר.

כדי להבין יותר טוב את מהלך העיבוד של הפונקציה FindResource, התבונן בתוכנית **FindRes**, שבתקליטור המצורף לספר זה. **FindRes** משתמשת ב-FindResource כדי לטעון **נתונים בינאריים גולמיים** (Raw Binary Data) לתוך מבנה (Structure). התוכנית **FindRes** משנה את קובץ המשאבים, את קובץ הכותר ואת קובץ התוכנית של קבצי **generic** המקוריים. קובץ המקור מכיל את הקוד הנוסף הבא, שפשוט מגדיר קבוצת ערכים הקסדצימליים:

```
TestData RCDATA DISCARDABLE
BEGIN
    0x0001,
    0x0002,
    0x0003
END
```

בקובץ הכותר מוגדר מבנה נתונים שהתוכנית קוראת את הנתונים לתוכו מתוך בלוק של TestData. היא עושה זאת כך:

```
typedef struct
{
   SHORT Value1;
   SHORT Value2;
   SHORT Value3;
} RESDATA;
```

לבסוף, קובץ התוכנית **FindRes** קורא את הנותנים לתוך המבנה בקטע הקוד של פריט התפריט !Test שנמצא בפונקציה WndProc, כך:

```
case IDM_TEST :
   {
     HRSRC hres = FindResource(hInst, "TestData", RT_RCDATA );
     if ( hres )
        {
           char szMsg[50];
           DWORD    size = SizeofResource( hInst, hres );
           HGLOBAL  hmem = LoadResource( hInst, hres );
           RESDATA* pmem = (RESDATA*)LockResource( hmem );
           wsprintf( szMsg, "Values loaded: %d, %d,
                        %d\nSize = %d", pmem->Value1,
                        pmem->Value2, pmem->Value3, size );
           MessageBox( hWnd, szMsg, lpszAppName, MB_OK );
        }
   }
   break;
```

אם התוכנית מוצאת נתוני משאבים תחת מילת המפתח TestData, אז התוכנית מחליטה על גודל הנתונים ושומרת אותו משתנה size. אחר כך משתמשת התוכנית במשתנה מסוג HGLOBAL כדי להקצות מקום בערימה (Space Off The Heap) לאחסנת הנתונים. לבסוף, התוכנית מאחסנת את הנתונים במופע (אובייקט או משתנה) של המבנה RESDATA. כשמהדרים ומפעילים את התוכנית ובוחרים באפשרות פריט התפריט !Test, יוצג על המסך את הערך בתוך תיבת הודעה.

סגנונות חלון

טבלה 12: הערכים האפשריים שהפרמטר **dwStyle** יכול לקבל כשיוצרים חלונות בתוכנית

סגנון	תיאור
WS_BORDER	יוצר חלון בעל גבול דק.
WS_CAPTION	יוצר חלון בעל שורת כותרת (כולל את הסגנון WS_BORDER).
WS_CHILD	יוצר חלון-בן. לא ניתן להשתמש בסגנון זה בשילוב עם הסגנון WS_POPUP.
WS_CHILDWINDOW	יוצר חלון כמו זה שנוצר על ידי WS_CHILD.
WS_CLIPCHILDREN	אינו כולל את השטח שחלונות הבן תופסים בעת שמתבצעת פעולת שרטוט בחלון האב. משתמשים בסגנון זה בעת יצירת חלון האב.
WS_CLIPSIBLINGS	גוזר חלונות בן יחסית זה לזה; כלומר, כשחלון מסוים מקבל ההודעה WM_PAINT, הסגנון WS_CLIPSIBLINGS גורם לגזירת כל שאר החלונות החופפים שמחוץ לשטח חלון הבן שהתוכנית משחזרת. אם התוכנית אינה מגדירה את הסגנון WS_CLIPSIBLINGS וחלונות הבן חופפים זה לזה, יכול להיות שבעת שרטוט בשטח הלקוח באחד מחלונות הבן, תמצא עצמך משרטט בשטח הלקוח של חלון שכן.
WS_DISABLED	יוצר חלון שמצבו ההתחלתי הוא בלתי פעיל. חלון במצב בלתי פעיל אינו יכול לקבל קלט מהמשתמש.
WS_DLGFRAME	יוצר חלון בעל גבול עם סגנון שמשתמשים בו בדרך כלל בתיבות דו-שיח. לחלון בעל סגנון כזה אין שורת כותרת.

סגנון	תיאור
WS_GROUP	מגדיר את הפקד הראשון בקבוצה של פקדים. הקבוצה מורכבת מהפקד הראשון הזה ושאר הפקדים שבקבוצה המוגדרים אחריו, עד לפקד הבא שמוגדר לפי סגנון WS_GROUP. בדרך כלל, לפקד הראשון בכל קבוצה יש סגנון WS_TABSTOP שמאפשר למשתמש לעבור בין הקבוצות. המשתמש יכול לעבור בין כל הפקדים שבקבוצה על ידי השימוש במקשי החיצים במקלדת.
WS_HSCROLL	יוצר חלון בעל פס גלילה אופקי.
WS_ICONIC	יוצר חלון שמצבו ההתחלתי מוקטן לסמל, כמו הסגנון WS_MINIMIZE.
WS_MAXIMIZE	יוצר חלון בגודל מקסימלי.
WS_MAXIMIZEBOX	יוצר חלון בעל לחצן הגדלה. התוכנית אינה יכולה לשלב את הסגנון WS_MAXIMIZEBOX עם הסגנון WS_EX_CONTEXTHELP. התוכנית גם חייבת להוסיף את הסגנון WS_SYSMENU כשהיא משתמשת בסגנון WS_MAXIMIZEBOX.
WS_MINIMIZE	יוצר חלון שמצבו ההתחלתי מוקטן לסמל, כמו בסגנון WS_ICONIC.
WS_MINIMIZEBOX	יוצר חלון בעל לחצן הקטנה. התוכנית אינה יכולה לשלב את הסגנון WS_MINIMIZEBOX עם הסגנון WS_EX_CONTEXTHELP. התוכנית גם חייבת להוסיף את הסגנון WS_SYSMENU כשהיא משתמשת בסגנון WS_MINIMIZEBOX.
WS_OVERLAPPED	יוצר חלון חופף. חלון חופף הוא בעל שורת כותרת וגבול, כמו בסגנון WS_TILED.
WS_OVERLAPPEDWINDOW	יוצר חלון חופף. החלון מכיל את הסגנונות WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX ו-WS_MAXIMIZEBOX.
WS_POPUP	יוצר חלון מוקפץ. אי אפשר להשתמש בסגנון זה עם הסגנון WS_CHILD.

סגנון	תיאור
WS_POPUPWINDOW	יוצר חלון מוקפץ. החלון מכיל את הסגנונות WS_BORDER, WS_POPUP ו-WS_SYSMENU. התוכנית חייבת לחבר את סגנונות החלון WS_CAPTION ו-WS_POPUPWINDOW כדי לגרום להצגת תפריט החלון.
WS_SIZEBOX	יוצר חלון בעל גבולות שמאפשרים שינוי הגודל שלו, כמו בסגנון WS_THICKFRAME.
WS_SYSMENU	יוצר חלון בעל תפריט מערכת בשורת הכותרת שלו. התוכנית חייבת להגדיר גם את הסגנון WS_CAPTION.
WS_TABSTOP	מגדיר פקד שיכול להיות ב**מיקוד הקלט** על ידי שימוש במקלדת (keyboard focus), כאשר המשתמש מקיש בטבלר (Tab key). ההקשה בטבלר גורמת לכך שמיקוד הקלט יעבור אל הפקד הבא בעל הסגנון WS_TABSTOP.
WS_THICKFRAME	יוצר חלון שיש לו גבולות שמאפשרים שינוי הגודל שלו, כמו בסגנון WS_SIZEBOX.
WS_TILED	יוצר חלון חופף. לחלון חופף יש שורת כותרת וגבול, כמו בסגנון WS_OVERLAPPED.
WS_TILEDWINDOW	יוצר חלון חופף. החלון מכיל את הסגנונות WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX ו-WS_MAXIMIZEBOX כמו בסגנון WS_OVERLAPPEDWINDOW.
WS_VISIBLE	מאפשר לראות את החלון מייד אחרי שנוצר.
WS_VSCROLL	יוצר חלון בעל פס גלילה אנכי.

טבלה 13: הערכים האפשריים שהפרמטר **dwStyle** יכול לקבל כשיוצרים **חלונות לחצן** (BUTTON windows)

סגנון	תיאור
BS_3STATE	יוצר לחצן כמו **תיבת סימון** (check box), שיכולה להיות גם במצב לא פעיל, בנוסף לשני המצבים של תיבת סימון רגילה, **מסומן** (checked) או **לא מסומן** (unchecked). משתמשים במצב לא פעיל כדי לסמן שהתוכנית עדיין לא החליטה על מצב תיבת הסימון.
BS_AUTO3STATE	יוצר לחצן כמו תיבת סימון בעל שלושה מצבים, ההבדל הוא שתיבה זו משנה את מצבה כשהמשתמש בוחר בה (מסמן אותה). מחזור המצבים הוא: מסומן, לא פעיל ולא מסומן.
BS_AUTOCHECKBOX	יוצר לחצן כמו תיבת סימון. ההבדל הוא בכך שתיבה זו משנה את מצב הסימון באופן אוטומטי בין מסומן לבין לא מסומן בכל פעם שהמשתמש בוחר בה.
BS_AUTORADIOBUTTON	יוצר לחצן אפשרויות אוטומטי, שדומה ללחצן אפשרויות רגיל (Option Button), אך נבדל ממנו בכך שכאשר המשתמש בוחר בלחצן, Windows קובעת את מצב הסימון שלו באופן אוטומטי כמסומן, ואוטומטית קובעת את מצב הסימון של שאר הלחצנים שבאותה קבוצה ללא מסומן.
BS_CHECKBOX	יוצר תיבת סימון קטנה וריקה עם טקסט. לפי ברירת המחדל, התוכנית מציגה את הטקסט מצד ימין של תיבת הסימון. להצגת הטקסט מצד שמאל של תיבת הסימון, צריך לחבר לסגנון זה את הסגנון BS_LEFTTEXT (או עם הסגנון השקול BS_RIGHTBUTTON).
BS_DEFPUSHBUTTON	יוצר לחצן רגיל שמתנהג כמו לחצן שנוצר עם הסגנון BS_PUSHBUTTON, ונוסף לו גבול שחור עבה. אם הלחצן נמצא בתיבת דו-שיח, אפשר לבחור בו על ידי הקשה על מקש Enter, גם אם הלחצן אינו נמצא **במיקוד הקלט** (input focus). סגנון זה שימושי כדי לאפשר המשתמש לבחור במהירות באפשרות ברירת המחדל.

סגנון	תיאור
BS_GROUPBOX	יוצר מלבן שבאמצעותו יכולה התוכנית להשתמש אחר כך כדי לתחום קבוצת פקדים. התוכנית מציגה את הטקסט הנלווה בסגנון זה בפינה השמאלית העליונה של המלבן. שם נרדף ל**תיבת הקבוצה** (GROUPBOX) הוא **מסגרת** (frame).
BS_LEFTTEXT	ממקם את הטקסט משמאל ללחצן האפשרויות, או משמאל לתיבת הסימון כאשר משלבים את הטקסט עם סגנון הלחצן או התיבה. דומה לסגנון BS_RIGHTBUTTON.
BS_OWNERDRAW	יוצר לחצן שמשורטט על ידי המשתמש (owner-drawn). חלון האב מקבל את ההודעה WM_MEASURITEM כש-Windows יוצרת את הלחצן ואת ההודעה WM_DRAWITEM כשאחד המרכיבים הוויזואליים של הלחצן משתנה. אין צורך לשלב את הסגנון הזה עם סגנון אחר.
BS_PUSHBUTTON	יוצר לחצן רגיל שמשגר הודעה WM_COMMAND לחלון האב כשהמשתמש בוחר בלחצן (לוחץ עליו).
BS_RADIOBUTTON	יוצר מעגל קטן עם טקסט. לפי ברירת המחדל, התוכנית מציגה את הטקסט מצד ימין של המעגל. להצגת הטקסט מצד שמאל של המעגל, צריך לשלב דגל זה עם הסגנון BS_LEFTTEXT (או עם הסגנון השקול BS_RIGHTBUTTON). בדרך כלל, לחצני האפשרויות משמשים ליצירת קבוצות של אפשרויות שיש קשר ביניהן, אך רק אחת מהן יכולה להתקיים בכל רגע נתון (mutually exclusive). לדוגמה, לחצני אפשרויות שמייצגים את התכונה יישור פיסקה במעבד תמלילים. הפיסקה יכולה להיות מיושרת לשמאל, לימין, לשני הצדדים, או לאמצע, אך אפשר לבחור רק באפשרות אחת בלבד, כדי להורות על יישור הפיסקה.
BS_USERBUTTON	סגנון זה הוא מוחלט, אך תואם עם גרסאות Windows של 16 סיביות. יישומים המבוססים על Win32 צריכים להשתמש ב-BS_OWNERDRAW.
BS_BITMAP	מציין שהלחצן מציג מפת סיביות (bitmap).
BS_BOTTOM	ממקם הטקסט בתחתית המלבן של הלחצן.
BS_CENTER	ממרכז את הטקסט אופקית במלבן של הלחצן.
BS_ICON	מציין שהלחצן מציג סמל.

תיאור	סגנון
מיישר לשמאל את הטקסט במלבן של הלחצן. אך אם הלחצן הוא תיבת סימון או לחצן אפשרויות שאין לו את הסגנון BS_RIGHTBUTTON, הטקסט ישאר מיושר לשמאל, אבל מצד ימין של תיבת הסימון או של לחצן האפשרויות.	BS_LEFT
המשך הזנת הטקסט לשורה הבאה, כשאורך מחרוזת הטקסט גדול יותר משורה בודדת של מלבן הלחצן.	BS_MULTILINE
מאפשר ללחצן לשלוח **הודעות שינוי מצב** (notification messages) BN_KILLFOCUS ,BN_DBLCLK ו-BN_SETFOUCS אל חלון האב של הלחצן. שים לב שלחצנים שולחים הודעת שינוי מצב BN_CLICKED ללא קשר אם יש ללחצן את הסגנון BS_NOTIFY.	BS_NOTIFY
גורם ללחצן (כמו תיבת סימון, תיבת סימון בעלת שלושה מצבים, או לחצן אפשרויות) להיראות ולפעול כמו לחצן רגיל (push button). הלחצן נראה בולט כאשר אינו לחוץ או במצב מסומן ושוקע בפנים כשנלחץ או כשמסמנים אותו.	BS_PUSHLIKE
מיישר לימין את הטקסט במלבן של הלחצן. אם הלחצן הוא תיבת סימון או לחצן אפשרויות שאין לו את הסגנון BS_RIGHTBUTTON, הטקסט נשאר מיושר לימין, אבל יהיה מצד ימין של תיבת הסימון או של לחצן האפשרויות.	BS_RIGHT
ממקם את המעגל של לחצן האפשרויות או את הריבוע של תיבת הסימון מצד ימין של מלבן הלחצן, כמו הסגנון BS_LEFTTEXT.	BS_RIGHTBUTTON
מציין שהלחצן יכול להציג טקסט.	BS_TEXT
ממקם את הטקסט למעלה במלבן של הלחצן.	BS_TOP
ממקם הטקסט באמצע (אנכית) המלבן של הלחצן.	BS_VCENTER

נספח ב - MFC

נספח זה נלקח מתוך הספר "המדריך השלם **Visual C++ 6**" בהוצאת הוד-עמי. הנספח מהווה תקציר של פרקים 1 עד 3 ולכן כמה נושאים עשויים להיראות לא שלמים.

הקדמה

גירסה 6.0 של **Visual C++** מראה כיצד מיקרוסופט ממשיכה להתמקד בטכנולוגיות האינטרנט ובתחום **COM** (Component Object Model, מודל עצם הרכיב), אשר מהווים נדבכים עיקריים של תפיסת **DNA** (Distributed interNet Application Architecture, ארכיטקטורת היישומים המבוזרים באינטרנט). נוסף לתמיכה ברכיבי תשתית אלה, Visual C++ 6.0 גם כוללת מספר רב של כלי עזר לשיפור יעילות העבודה, כמו למשל Edit And Continue, IntelliSense, AutoComplete, וכלים רבים אחרים לתכנות מתקדם. כלים ורכיבי התכנות השונים מציגים גירסה זו של Visual C++ בדרגה גבוהה יותר ממה שהכרת עד כה. כמחברים עשינו מאמצים רבים להבטיח שספר זה יתמוך ברצונך לשפר את יכולתך בשפת Visual C++ ובאפשרויותיה המגוונות והמתקדמות, כמו גם ניצול הטכנולוגיות החדשות שפורטו והוסברו בהרחבה בספר שלפניך.

MFC, ATL ו-WFC - האם MFC מת?

בוודאי שלא! מאז שמיקרוסופט שחררה את **ATL** (Active Template Library, ספריית התבניות האקטיבית) כחלק של שפת התכנות Visual C++, טענו מפתחים רבים לסביבת Windows שמיקרוסופט נוטשת את **MFC** (Microsoft Foundation Class Library, ספריית מחלקות התשתית) ועליהם להפנות את מאמצי התכנות שלהם לסביבות עבודה חדשות, כמו למשל ATL. לאחרונה שחררה מיקרוסופט ספריית מחלקות חדשה בשם **WFC** (Windows Foundation Classes, מחלקות התשתית של Windows) המיועדת למפתחי Java, אשר באופן בלתי צפוי תמכה בשמועות אודות "מותה של MFC".

השמועות על הכחדת MFC אינן מבוססות כלל ועיקר. הגירסה החדשה Visual C++ 6.0 מוסיפה פונקציונליות משמעותית הן ל-MFC והן ל-ATL, והדבר מצביע על כך שמיקרוסופט תמשיך להקדיש תשומת לב זהה לשתי מערכות אלו. חלק מהבעיה בכך שיעדי העיצוב של ספריות אלו אינם מוצגים לעיתים בבהירות מספקת, ולכן גם אינם מובנים למפתחים. MFC תוכננה ועוצבה להיות ספריית מחלקות גדולה לפיתוח יישומי

חלונות מתוחכמים ועתירי גרפיקה. ATL תוכננה ועוצבה להקל על פיתוח עצמי COM פשוטים ופקדי ActiveX. כל אחד מיעדים אלה תרם לפיתוח ספריה בעלת אופי שונה לתמיכה במאמצי הפיתוח של מפתחי יישומי Windows.

טעות נפוצה אחרת היא במחשבה ש-MFC ו-ATL סותרות זו את זו, או שכל אחת מספריות אלו מבטלת את הצורך להשתמש בספריה האחרת. זה בהחלט לא נכון! למעשה, קל מאוד ליצור עצמי COM מבוססי-ATL אשר משתמשים ב-MFC. הבעיה היחידה, לכאורה, היא שמכיון שמפתחים רבים בוחרים בדרך כלל ב-ATL לעבודות הקלות והפשוטות יותר, השימוש ב-MFC, ספריית המחלקות ה"עשירה" וה"כבדה" נראה להם מנוגד לסיבה שבחרו ב-ATL מלכתחילה. ייתכן שקושי זה מתייחס למפתחים מסוימים, אך אין זה גורם לכך שהספריה ATL שוללת את MFC, או להיפך.

ATL אמנם אינה מחליפה את MFC, אך אנו סבורים שחשוב לדעת ששתי ספריות אלו הן חלק בלתי נפרד משפת ++Visual C.

ספריית התבניות של ActiveX (או בקיצור, **ATL**) היא כלי נפרד מ-MFC, שמיועד ליצירת פקדי ActiveX. ניתן ליצור פקדים אלה בעזרת הספריה MFC או ATL, אך פקדי ATL קטנים יותר וקל יותר לטעון אותם באינטרנט.

השם **++Visual C** עשוי להטעות. ניתן לחשוב שמדובר במערכת תכנות חזותית (Visual) טהורה, בדומה ל-Visual Basic של מיקרוסופט, ואכן זו התחושה בתחילת העבודה. בפועל מתברר, שיש צורך לקרוא ולכתוב פקודות בשפת **++C**. אשפי ++Visual C אכן חוסכים זמן ומשפרים את הדיוק, אך על המתכנת להבין את הקוד שהם יוצרים, ובסופו של דבר עליו להכיר גם את מבנה ספריית מחלקות התשתית של מיקרוסופט - **MFC** (Microsoft Foundation Class) ואת דרכי הפעולה הפנימיות של מערכת ההפעלה Windows.

Windows 9x לעומת Windows NT

ניתן לעבוד עם גירסה 6.0 של ++Visual C תחת Windows 9x או Windows NT בגירסה 4.0 ומעלה, שממשק המשתמש שלהן זהה. אני ממליץ לעבוד תחת Windows NT, אשר יציבותו מאפשרת לו לפעול במשך חודשים רבים בלי לאתחל את המחשב אפילו פעם אחת. אם תשתמש בממשק התכנות MFC בלבד, התוכניות שתהדר תפעלנה תחת Windows 9x ו-Windows NT; אך זכור שכאשר התוכנית מכילה קריאות לפונקציות מבוססות Win32 שמנצלות רכיבים מסוימים של Windows NT או של Windows 98, היא תפעל תחת מערכת הפעלה זו בלבד.

מתקדמים שלב נוסף עם Windows: סרגלי הצד המיועדים למתכנתי Win32

על מנת להבין את כל הפרטים, התחבולות והרכיבים החדשים של Win32, עליך להבין את ממשק תכנות היישומים (API) Win32 ואת יחסי הגומלין שלו עם ספריית MFC, ובמיוחד את כל מה שקשור במנגנון שיגור המסרים של Windows והבנת תפקיד המחלקות של **Windows**.

כשמציגים תוכנית C מקובלת בעזרת ממשק תכנות היישומים של Windows, כל שתזדקק לו הוא קוד המקור. מערכת יישומי ספריית MFC מסבכת מעט את העניין. רוב קטעי הקוד בשפת ++C נוצרים על ידי אשף היישומים (AppWizard) והמשאבים שמקורם בעורכי המשאבים (Resource Editors). הדוגמאות המופיעות בהמשך, כוללות הוראות מפורטות כיצד ליצור ולהתאים את קוד המקור בעזרת הכלים השונים. מומלץ מאוד לעיין בהוראות אלו בעבודה עם הדוגמאות הראשונות (יש להקליד מעט מאוד פקודות).

למתכנתי Win32: Unicode

עד לאחרונה, תוכניות מבוססות Windows ניצלו את מערך תווי ANSI, שכולל 256 תווים שכל אחד מהם תופס בית אחד. מפתחים שמייעדים את תוכניותיהם לשוק התוכנה האסיאתי (יפן, למשל), עוברים למערך התווים Unicode, הכולל 65,536 תווים שכל אחד מהם מורכב מ-2 בתים (תו מורחב). קיים מערך אחר של תווים כפולי-בתים בשם DBCS, הכולל תווים בעלי בית יחיד וכאלה של 2 בתים, אך הוא אינו מקובל.

ספריית MFC וספריית זמן ריצה תומכות ביישומי Unicode. אם תגדיר את הקבוע _UNICODE ותפעל על פי השלבים המתוארים בעזרה המקוונת, כל משתני התווים ומחרוזות הקבועים יהיו "רחבים" (Wide), והמהדר ייצור קריאות לגרסאות מורחבות-תווים (Wide-Characters) של פונקציות Win32. מנגנון זה מבוסס על ההנחה שתנצל פקודות מאקרו מוגדרות בעת הכרזה על מצביעי תווים (Character Pointers) ומערכים. לדוגמה, TCHAR ו-_T.

עם זאת, כאשר תנסה להריץ את יישומי MFC Unicode שכתבת תחת Windows 9x תיתקל בקשיים, מכיון ש-Windows אינה מכילה רכיבי תמיכה ב-Unicode. Windows 9x כוללת גרסאות מורחבות תווים של פונקציות Win32, אולם הן מחזירות קוד שגיאה בלבד. Windows NT, לעומת זאת, תומכת ב-Unicode וכוללת שתי גרסאות של פונקציות Win32 שמטפלות בתווים. אם תקרא לגרסאות של בית בודד, Windows NT תבצע את ההמרה הדרושה לתווים מורחבים ולהיפך.

✍ **הערה:** תוכניות ההדגמה בספר "המדריך השלם Visual C++ 6" אינן מותאמות ל-Unicode. כל התוכניות מנצלות סוגים וקבועי מחרוזות יחידי-בית, כגון char, ואינן מגדירות UNICODE. אם תריץ את הדוגמאות תחת WindowsNT, מערכת ההפעלה תבצע את ההמרה הדרושה מבית אחד לשני בתים כנדרש. אם תריץ אותן תחת Windows 9x, הממשק הינו של תווי בית אחד בלבד.

קיים תחום אחד בו תיאלץ בכל זאת להתמודד עם תווים כפולי-בתים: זהו COM. פונקציות COM שאינן שייכות ל-MFC (להוציא פונקציות DAO) שכוללות פרמטרים של מחרוזות ותווים, מחייבות שימוש בתווים כפולים (OLECHAR). אם תכתוב תוכנית שאינה Unicode, תאלץ לבצע את ההמרות בעצמך בעזרת המחלקה CString של MFC ופקודות מאקרו נוספות של MFC.

אם ברצונך לכתוב יישומי Unicode, קרא את הפרק העוסק ב-Unicode בספרו של ג'יפרי ריכטר, Advanced Windows. חשוב גם שתקרא את הפרקים העוסקים ב-Unicode בעזרה המקוונת של Visual C++.

Windows ו- Visual C++ של מיקרוסופט

רבות נכתב כבר על מערכת ההפעלה Windows של מיקרוסופט ועל יתרונות ממשק המשתמש הגרפי (GUI) שלה. עכשיו, נסכם את מודל התכנות של Windows (ובעיקר - Win32) ונסביר כיצד פועלים רכיבי Visual C++ זה עם זה כדי לאפשר כתיבת תוכניות לסביבת Windows. תוך קריאת הפרק, ייתכן שתפגוש נתונים ועובדות חדשות אודות Windows, שלא הכרת קודם לכן.

מודל התכנות של Windows

יהיה כלי התכנות אשר יהיה, תכנות בסביבת Windows שונה מהתכנות המיושן מוכוון-אצווה (Batch-Oriented) או מוכוון-תנועה (Transaction-Oriented). לצורך צעדים ראשונים בתחום זה, עליך להכיר מספר עקרונות בסיסיים של Windows. כמסגרת התייחסות, ננצל את מודל התכנות הידוע של MS-DOS. גם אם אינך מתכנת לסביבת MS-DOS פשוטה, הנושא בוודאי מוכר לך.

טיפול בהודעות

כשכותבים יישום מבוסס DOS בשפת C, הדרישה המוחלטת היחידה היא ליצור פונקציה בשם **main**. מערכת ההפעלה קוראת ל-main כשהמשתמש מריץ את התוכנית, ומאותו שלב והלאה התוכנית יכולה להיכתב במבנה תכנות כלשהו. אם התוכנית מקבלת קלט באמצעות המקלדת, או מנצלת בצורה כלשהי את שירותי מערכת ההפעלה, היא קוראת לפונקציות המתאימות, כגון getchar, או מנצלת את ספריית החלונות מבוססת-התווים.

כשמערכת ההפעלה Windows מפעילה תוכנית, היא קוראת לפונקציה **WinMain** של התוכנית. היישום חייב להכיל פונקציה בשם WinMain, שמבצעת מספר משימות מוגדרות. המשימה החשובה ביותר היא יצירת החלון הראשי של היישום, אשר כוללת קוד נפרד לטיפול בהודעות ששולחת מערכת ההפעלה. ההבדל המובהק בין תוכנית שנכתבה עבור MS-DOS לתוכנית שמיועדת לסביבת Windows, שתוכנית מבוססת DOS קוראת למערכת ההפעלה כדי לקבל קלט מהמשתמש, בעוד שתוכנית מבוססת Windows מטפלת בקלט המשתמש באמצעות הודעות שמגיעות ממערכת ההפעלה.

✍ **הערה:** סביבות תכנות חלונאיות רבות, ובכלל זה Visual C++ 6.0 של מיקרוסופט עם ספריית MFC גירסה 6.0, מפשטות את תהליך התכנות על ידי הסתרת הפונקציה WinMain והַבְנָיַית (Structuring) תהליך הטיפול בהודעות. כשאתה מנצל את ספריית התשתיות MFC, אינך צריך לכתוב פונקציה WinMain, אך חיוני שתבין את הזיקה בין מערכת ההפעלה והתוכניות שאתה כותב.

רוב ההודעות של Windows מוגדרות ויישימות לכל התוכניות. לדוגמה, ההודעה WM_CREATE נשלחת כאשר נוצר החלון, ההודעה WM_LBUTTONDOWN נשלחת כאשר המשתמש לוחץ על הלחצן השמאלי בעכבר, ההודעה WM_CHAR נשלחת כאשר מקישים על מקש תו כלשהו וההודעה WM_CLOSE נשלחת כשסוגרים חלון. לכל ההודעות יש שני פרמטרים של 32 סיביות, אשר מעבירים מידע: קואורדינטות הסמן, קוד מקש ועוד. מערכת ההפעלה שולחת הודעות WM_COMMAND אל החלון המתאים, בתגובה לאפשרות התפריט שהמשתמש בוחר, לחיצות על לחצני הדו-שיח וכו׳. פרמטרי הודעת הפקודה משתנים בהתאם לעיצוב תפריט החלון. תוכל להגדיר הודעות משלך, שהתוכנית תשלח לחלון כלשהו בשולחן העבודה. הודעות אלו, שמוגדרות על ידי המשתמש, גורמות ל- ++C להידמות במידה מסוימת ל-Smalltalk.

בשלב זה אינך צריך להבין מהו הקשר בין הודעות אלו לתוכנית שכתבת. זו משימתה של מערכת היישום. יחד עם זאת, הייה ער לעובדה שעיבוד ההודעות של Windows מאלץ את התוכנית שלך להיות מורכבת מאוד. אל תנסה לגרום לתוכניות החלונאיות שלך להיראות כמו התוכניות שכתבת פעם לסביבת MS-DOS. למד בעיון את הדוגמאות שבספר והתכונן לפתוח ״דף חדש״ בסגנון התכנות.

ממשק ההתקן הגרפי של Windows

תוכניות MS-DOS רבות שלחו את נתוני הפלט שלהן ישירות לזיכרון הווידאו וליציאת התקשורת של המדפסת. החיסרון בשיטה זו, הוא הצורך לספק תוכנת דרייבר נפרדת לכל כרטיס וידאו ודגם של מדפסת. Windows כוללת רובד הַפְשָטָה (Abstraction) שנקרא ממשק התקן גרפי - Graphics Device Interface) GDI). מערכת ההפעלה מספקת את הדרייברים של הווידאו והמדפסת, ולכן התוכנית אינה חייבת לזהות את ההתקנים האלה הקשורים למחשב. במקום שהתוכנית ״תדבר״ עם ההתקן ברמת החומרה, היא קוראת לפונקציות GDI שפונות למבנה נתונים בשם הֶקשר ההתקן (Device Context). Windows ממפה את מבנה הקשר ההתקן להתקן פיסי ושולחת אליו את פקודות הקלט-פלט המתאימות. קצב העבודה, או הגישה, באמצעות ממשק GDI זהה כמעט לזה של גישה ישירה לכרטיס הווידאו וכך, יישומים חלונאיים שונים יכולים לשתף ביניהם את צג המחשב ללא קושי.

תכנות משותף-משאבים (Resource Based Programming)

לתכנות מונַע-אירועים (Event-Driven Programming) בסביבת MS-DOS, עליך לקודד את הנתונים כקבועי אתחול, או להקצות לתוכנית שכתבת קבצי נתונים נפרדים. בתכנות לסביבת Windows מאחסנים נתונים בקובץ נתונים באמצעות מספר תבניות מוגדרות מראש. תוכנית הקישור (Linker) מצרפת את קובץ המשאבים הבינארי אל קובץ הפלט שהפיק מהדר ++C ויוצרת על ידי כך תוכנית הפעלה (Executable Program). קבצי משאבים יכולים להכיל מפות סיביות (Bit Maps), סמלים (Icons), תבניות של תיבות דו-שיח ומחרוזות (Strings). הם יכולים להכיל אפילו תבניות משאבים שהוגדרו במיוחד למטרה זו.

את התוכנית כותבים באמצעות עורך טקסט, אך לעריכת המשאבים משתמשים בדרך כלל בכלי **WYSIWYG** (What You See Is What You Get, או בעברית - מה שאתה רואה הוא מה שתקבל). אם תעצב תיבת דו-שיח, למשל, תוכל לבחור מרכיבים (לחצנים, תיבות רשימה ועוד) מתוך מערך סמלים, שנקרא לוח פקדים (Control Palette), ולהציבם במקום הרצוי בעזרת העכבר. Visual C++ 6.0 היא סביבת הפיתוח המשולבת של Visual C++ ומכילה עורכי משאבים גרפיים לכל תבניות המשאבים המקובלות.

ניהול הזיכרון

ניהול הזיכרון הולך ונעשה קל ופשוט בכל גירסה חדשה של Windows. שמעת בוודאי סיפורי אימים על אודות נעילת ידיות זיכרון (Memory Handles), thunks ו-burgermasters, אל דאגה. כל זה שייך להיסטוריה. כל שעליך לעשות הוא להקצות את הזיכרון הדרוש לך, ו-Windows תטפל בפרטים הקטנים.

ספריות dll

בסביבת מערכת ההפעלה MS-DOS, כל מודולי העצמים של התוכנית נקשרים בצורה סטטית בתהליך בניית קובץ ההפעלה. Windows מאפשרת קישור דינמי (Dynamic Linking), שמשמעותו טעינה וקישור בזמן ריצה של ספריות שנבנו במיוחד למטרה זו. יישומים רבים יכולים לשתף את ספריות הקישור הדינמי, או כפי שהן נקראות - ספריות **DLL** (Dynamic Link Libraries), דבר שחוסך משאבי זיכרון ומקום בדיסק. הקישור הדינמי מגדיל את המודולריות של התוכנית, מכיון שניתן להדר את הספריות בנפרד מהרצתן לניסוי, למשל.

מפתחי שפת התכנות יצרו תחילה ספריות DLL לתכנות בשפת C, אך שפת C++ יצרה קשיים מסוימים בעניין זה. מפתחי MFC (מחלקת התשתיות של מיקרוסופט) הצליחו לשלב את כל מחלקות סביבת היישום - Application Framework Classes) AFC) למספר מצומצם של ספריות DLL מוגדרות מראש. משמעות הדבר, שניתן לקשר את מחלקות סביבת היישום אל היישום עצמו באופן סטטי או דינמי. בנוסף, תוכל ליצור ספריות DLL כהרחבה לספריות MFC הקיימות.

ממשק תכנות היישומים Win32

כשהחל השימוש במערכת ההפעלה Windows לשוק כתבו המתכנתים תוכניות בשפת C עבור סביבת ממשק תכנות היישומים - Application Programming Interface) API) בארכיטקטורת Win16. כשארכיטקטורת Win32 נכנסה לשוק, הוברר שכתיבת יישומים מבוססי 32 סיביות מחייבת לעבוד עם ממשק התכנות Win32, באופן ישיר או עקיף. לרוב הפונקציות שמבוססות על Win16 יש פונקציות מקבילות שמבוססות על Win32, אך רוב הפרמטרים שונים: פרמטרים של 16 סיביות מוחלפים לעיתים קרובות על ידי פרמטרים של 32 סיביות. ממשק התכנות Win32 כולל פונקציות רבות חדשות שמבצעות פעולות כגון קריאה וכתיבה לדיסק, שהיו בעבר באחריות מערכת ההפעלה MS-DOS. בגרסאות 16 סיביות של Visual C++, ההבדלים בין ממשקי התכנות היו שקופים במידה רבה למתכנתי MFC, מכיון שאלה כתבו תוכניות בהתאם לתקן MFC, אשר תוכנן לאפשר עבודה בשתי הארכיטקטורות.

רכיבי שפת התכנות ++Visual C

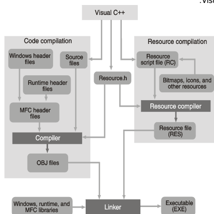
שפת ++Visual C של מיקרוסופט כוללת למעשה שתי מערכות פיתוח של יישומי Windows המוכללות במוצר אחד. אם תבחר, תוכל לפתח יישומי C לסביבת Windows באמצעות ממשק התכנות Win32 בלבד. לשם כך תוכל לנצל כלי ++Visual C אחרים, כולל עורכי משאבים שיקלו עליך במטלות תכנות בסיסי (Low-Level Programming).

++Visual C כוללת גם את ספריית התבניות של **ActiveX** - ATL (ActiveX Template Library), שבעזרתה ניתן לפתח פקדי ActiveX עבור יישומי אינטרנט. תכנות ATL שונה הן מתכנות בשפת C בארכיטקטורת 32 סיביות והן מתכנות MFC, ולאמיתו של דבר הוא כה מורכב, שראוי להקדיש לו ספר נפרד.

בהמשך נדגים תכנות בשפת ++C בסביבת ספריית MFC אשר שייכת לכלי הפיתוח של ++Visual C. במהלך עבודתך תנצל מחלקות ++C שמתועדות במסגרת MFC, וגם כלי ++Visual C ייחודיים לסביבת היישום, כגון AppWizard ו-ClassWizard.

✍ **הערה:** ממשק התכנות של ספריית MFC אינו מנתק אותך לחלוטין מפונקציות Win32. למעשה, כמעט תמיד יהיה צורך בקריאות ישירות לפונקציות אלו מתוך תוכניות ספריית MFC.

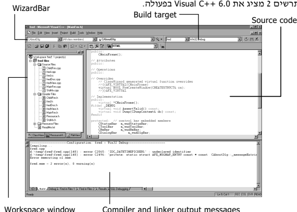
סקירה מהירה של רכיבי ++Visual C תסייע לך להתמצא בתוכנה זו בטרם נעבור לסביבת היישומים. תרשים 1 מציג מבט על תהליך הבנייה של יישום (Build Process) בשפת ++Visual C.



תרשים 1 : תהליך בניית יישום בשפת ++Visual C

Microsoft Visual C++ 6.0 ותהליך הבנייה של יישום

Visual C++ 6.0 הוא סביבת פיתוח משולבת - **IDE** (Integrated Development Environment) שמתקיימת בתוך סביבת Windows, אשר משותפת ל- Visual C++, Microsoft Visual J++, Microsoft Visual Basic ומוצרים נוספים. סביבת פיתוח משולבת זו, IDE, עברה דרך ארוכה מאז היות Visual Workbench אשר התבססה על QuickC עבור Windows. חלונות, סרגלי כלים מותאמים ועורך מותאם שמפעיל פקודות מאקרו, כל אלה מהווים כיום חלק מסביבת Visual Studio. מערכת העזרה המקוונת (כעת מוכללת בתוך התצוגה של ספריית MSDN) פועלת בצורה דומה לדפדפן Web. תרשים 2 מציג את Visual C++ 6.0 בפעולה.



תרשים 2 : חלון Visual C++ 6.0

אם הפעלת גרסאות מוקדמות של Visual C++ או את IDE של בורלנד, אתה יודע כיצד פועלת Visual C++ 6.0. אם אלה צעדיך הראשונים בעבודה עם IDE, עליך להכיר תחילה את המושג פרויקט. פרויקט (Project) הוא אוסף של קבצים שיש ביניהם קשרי גומלין, אשר עוברים תהליך הידור וקישור ליצירת תוכניות הפעלה או ספריות DLL של Windows. קבצי המקור של פרויקט מוגדר נשמרים בתת-ספריה מוגדרת. הפרויקט תלוי גם בקבצים רבים שנמצאים מחוץ לתת-הספריה שהוקצתה עבורו, כגון קבצי הכללה (Include Files) וקבצי ספריה (Library Files).

מתכנתים מנוסים מכירים את קבצי make. קובץ **make** מכיל אפשרויות שונות של המהדר ושל תוכנית הקישור, ואת כל המידע אודות יחסי הגומלין בין קבצי המקור. קוד מקור זקוק לקבצי הכללה מסוימים, קובץ הפעלה זקוק למודולי עצמים וספריות מסוימות וכן הלאה. תוכנית **make** קוראת את קובץ make ולאחר מכן מפעילה את המהדר, assembler, מהדר המשאבים ותוכנית הקישור ליצירת הפלט הסופי, בדרך

ככל יהיה זה קובץ הפעלה (Executable File). תוכנית make מנצלת את כללי ההסקה (Inference) המובנים שלה שמנחים אותה, למשל, להפעיל את המהדר כדי ליצור קובץ OBJ שמבוסס על קובץ CPP מוגדר.

בפרויקט של Visual C++ 6.0 אין קבצי make כלשהם (בעלי הסיומת MAK), אלא אם תציין במפורש שאתה רוצה להפיק קובץ כזה. קובץ פרויקט (Project File) בתבנית טקסט (שהסיומת שלו DSP) ממלא מטרה זהה. קובץ שטח עבודה (Workspace File) מבוסס-טקסט (שהסיומת שלו DSW) כולל הגדרה לכל פרויקט בשטח העבודה. ניתן להגדיר פרויקטים רבים בשטח עבודה משותף, אך בכל הדוגמאות המופיעות בספר תמצא פרויקט אחד בלבד בשטח עבודה נתון. כדי לעבוד על פרויקט קיים, עליך לומר ל- Developer C++ לפתוח את קובץ DSW תחילה.

Visual C++ יוצרת מספר קבצי ביניים. הטבלה שלפניך מציגה את הקבצים שנוצרים בשטח העבודה:

טבלה 1: תיאור הקבצים שנוצרים בשטח העבודה

סיומת הקובץ	תיאור
APS	תמיכה בתצוגת המשאבים (ResourceView)
BSC	קובץ מידע לדפדפן
CLW	תמיכה ב-אשף המחלקות (ClassWizard)
DEP	קובץ תלוי
DSP	קובץ פרויקט*
DSW	קובץ שטח עבודה*
MAK	קובץ make חיצוני
NCB	תמיכה בתצוגת מחלקות (ClassView)
OPT	שמירת תצורת שטח העבודה
PLG	בניית קובץ יומן

* אין למחוק או לערוך בעורך טקסט

עורכי המשאבים - שטח העבודה של ResourceView

כשבוחרים בכרטיסיה של תצוגת המשאבים - ResourceView - בחלון שטח העבודה של Visual C++, אפשר אחר כך לבחור משאב כלשהו מתוך רשימת המשאבים המוצגת ולערוך אותו. החלון הראשי מכיל עורך משאבים (Resource Editor) שמתאים לסוג המשאב שנבחר. החלון יכול גם להכיל עורך WYSIWYG עבור תפריטים, או עורך גרפי רב-עוצמה עבור תיבות דו-שיח. הוא מכיל בדרך כלל כלי עריכה לסמלים, מפות סיביות ומחרוזות. עורך תיבות הדו-שיח מאפשר לשלב פקדי ActiveX נוסף על הפקדים הרגילים של Windows וגם את פקדי Windows המשותפים החדשים.

כל פרויקט כולל בדרך כלל קובץ אחד של תסריט משאבים מבוסס-טקסט, ובקצרה - קובץ תסריט המשאבים (בעל סיומת RC), אשר מתאר את רכיבי הפרויקט השונים כגון התפריט, תיבת דו-שיח, מחרוזות ומשאבי קיצור (accelerator resources). קובץ RC כולל גם משפט **#include** שתפקידו לייבא משאבים מתת-ספריות אחרות. משאבים אלה כוללים פריטים ייחודיים לפרויקט, כגון מפות סיביות (BMP), קבצי סמלים (ICON) ומשאבים משותפים לכל תוכניות ++Visual C, כמו מחרוזות של הודעות שגיאה. לא מומלץ לערוך קבצי RC שלא באמצעות עורכי המשאבים. עורכים אלה מסוגלים גם לערוך קבצי EXE ו-DLL, כך שתוכל לנצל את לוח הגזירים (Clipboard) ל"חטיפת" משאבים, כגון מפות סיביות וסמלים מיישומי Windows אחרים.

מהדר לתוכניות ++C/C

מהדר ++Visual C מסוגל לטפל בקוד שנכתב בשפת C או ++C. המהדר קובע באיזו שפה כתובה התוכנית על פי הסיומת של שם קובץ המקור: הסיומת C מציינת קובץ בשפת C, והסיומת CPP או CXX מעידה על קובץ מקור בשפת ++C. המהדר עומד בכל דרישות ANSI, כולל ההמלצות העדכניות של קבוצת העבודה לספריות ++C והרחבות של מיקרוסופט. גירסה 6.0 של ++Visual C תומכת באופן מלא בתבניות (Templates), בחריגים (Exceptions) ובזיהוי סוג בזמן ריצה (Runtime Identification). המהדר כולל גם את ספריית התבניות הסטנדרטיות - **STL** (Standard Template Library) החדשה של ++C, למרות שזו אינה כלולה בספריית MFC.

עורך קוד מקור

Visual C++ 6.0 כולל עורך קוד מקור מורכב שתומך בהרבה יתרונות (רכיבים חדשים) כמו צביעת תחביר דינמית, טאבים באופן אוטומטי, כריכות לוח מקשים עבור מיגוון של עורכים פופולריים (כמו VI ו-EMACS), והדפסה פשוטה. ב- Visual C++ 6.0 התווסף יתרון נוסף שנקרא השלמה אוטומטית (AutoComplete). אם השתמשת באחד מהמוצרים של Microsoft Office או ב- Microsoft Visual Basic, הטכנולוגיה הזו כנראה כבר מוכרת לך. באמצעות רכיב זה של Visual C++ 6.0, כל שעליך לעשות הוא להקיש את ההתחלה של אחד מקבועי התכנות והעורך יציע לך רשימה של המשכים שמהם באפשרותך לבחור אחד. רכיב זה שימושי מאוד כאשר אתה עובד עם עצם של ++C ושכחת את השם המדויק של פונקציה חברה או של נתון חבר – כולם נמצאים עבורך בתוך הרשימה. תודות ליכולת זו אינך צריך לזכור אלפי פונקציות API של Win32 או להיות תלוי במערכת העזרה המקוונת.

מהדר המשאבים

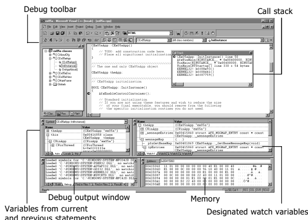
מהדר המשאבים (Resource Compiler) של ++Visual C קורא קובץ תסריט (RC) מבוסס ASCII שנוצר על ידי עורך משאבים כלשהו, ויוצר קובץ בינארי עבור תוכנית הקישור (Linker).

תוכנית הקישור

תוכנית הקישור (Linker) קוראת קבצי OBJ וקבצי RES שיצרו מהדר ++C/C ומהדר המשאבים, וניגשת אל קבצי LIB כדי להביא מהם קוד MFC, קוד ספריית זמן ריצה וקוד Windows. לאחר מכן, תוכנית הקישור יוצרת את קובץ ההפעלה, שהסיומת שלו היא EXE. אפשרות הקישור המדורגת מצמצמת את זמן הביצוע במקרה שנערכו שינויים מזעריים בקבצי קוד המקור. קבצי הכותר (Header Files) של MFC כוללים משפטי **pragma#** (הוראות מיוחדות למהדר), שתפקידם לציין את הספריה שמכילה את הקבצים הדרושים. הדבר פוטר את המתכנת מלציין במפורש איזו ספריה צריך לקרוא.

תוכנית ניפוי שגיאות

אם התוכנית שכתבת פועלת כבר בפעם הראשונה, אינך זקוק לתוכנית ניפוי שגיאות (Debugger) כלשהי. בדרך כלל אין הדברים כך, ולכן תזדקק לתוכנית זו מפעם לפעם. תוכנית הניפוי של ++Visual C עברה שיפורים מתמידים, אך אין ביכולתה עדיין לתקן בפועל את שגיאות התכנות. התוכנית פועלת במשותף עם ++Visual C כדי להבטיח שנקודות העצירה (Breakpoints) תישמרנה בדיסק. לחצני סרגל הכלים מאפשרים להכניס ולבטל נקודות עצירה במהלך התוכנית, והם שולטים בהפעלת התוכנית בשיטת צעד-אחר-צעד (Single-Step).



תרשים 3 : חלון תוכנית ניפוי השגיאות של ++C

תרשים 3 מתאר את רכיב ניפוי השגיאות של ++Visual C בפעולה. שים לב שתיבות המשתנים והתצוגה יכולים להרחיב מצביע עצם (Object Pointer) כדי שיציג את כל איברי הנתונים (Data Members) של המחלקה הנגזרת ושל מחלקת הבסיס. אם תציב את הסמן על משתנה פשוט, תוכנית הניפוי תציג את ערכו הנוכחי בתיבה קטנה בתוך החלון. אם ברצונך לאתר שגיאות בתוכנית, עליך לבחור את האפשרויות המתאימות של המהדר ותוכנית הקישור, כך שיפיקו נתוני ניפוי שגיאות.

Visual C++ 6.0 הוסיפה עיוות מסוים לתיקון שגיאות עם יכולת של עריכה והמשכה (Edit and Continue). היכולת עריכה והמשכה מאפשרת לך לתקן יישומים, לשנות את התוכנית, לשנות את היישום, ואז להמשיך את התיקונים עם הקוד החדש. יכולת זו תוריד בצורה משמעותית את כמות הזמן שאתה מאבד עבור התיקונים, מפני שאין לך יותר צורך לעזוב את תוכנית ניפוי השגיאות, להדר אותה שוב, ואחר כך לנפות שגיאות שוב. כדי להשתמש ביכולת זו, ערוך את התוכנית כאשר מנפה התוכנית עובד, ואחר כך לחץ על הלחצן המשך. Visual C++ 6.0 מהדרת את השינויים באופן אוטומטי ומתחילה מחדש את מנפה התוכניות עבורך.

אשף היישומים - AppWizard

AppWizard הוא כלי שיוצר שלד עובד של יישום Windows שניתן להפעילו וכולל רכיבים, שמות של מחלקות ושמות קבצי קוד מקור, שאותם מגדירים באמצעות תיבות דו-שיח מיוחדות. AppWizard ישמש אותך רבות כשתתרגל את הדוגמאות המוצגות בספר. אל תבלבל בין AppWizard למחוללי קוד ישנים, שמפיקים את כל הקוד הדרוש ליישום. הקוד שיוצר AppWizard הוא מינימלי בלבד; יכולת התפקוד של היישום נובעת מהקוד שנמצא במחלקות הבסיס של סביבת היישום. כל ש-AppWizard עושה הוא לסייע לך בצעדיך הראשונים בכתיבת יישום חדש.

מתכנתים מנוסים יכולים ליצור לעצמם AppWizard שמותאם לצרכיהם. חברת מיקרוסופט חשפה את המערכת מבוססת-מאקרו שלה ליצירת פרויקטים. אם צוות הפיתוח צריך לפתח פרויקטים רבים באמצעות ממשק תקשורת, תוכל לבנות אשף מיוחד שיהפוך תהליך זה לאוטומטי.

אשף המחלקה - CLASSWIZARD

אשף המחלקה ClassWizard הוא תוכנית (מופיע כ-DLL) שניתן להגיע אליה באמצעות תפריט View של ++Visual C. האשף משחרר אותך מהטרדה הכרוכה בתחזוקת קוד המחלקה של תוכנית ++Visual C. אתה זקוק למחלקה חדשה, פונקציה וירטואלית חדשה או פונקציה לטיפול בהודעה? האשף ייצור עבורך את התבנית הכללית, כלומר גוף הפונקציה, ובמידת הצורך גם את הקוד הדרוש לקישור הודעת Windows לפונקציה שייצר. ClassWizard מסוגל לעדכן קוד מחלקה שייצרת, וכך הוא משחרר אותך מבעיות התחזוקה הטיפוסיות למחוללי קוד רגילים. תרשים 2 מציג חלק מרכיבי ClassWizard שניתן להפעילם באמצעות סרגל הכלים של Developer Studio.

דפדפן המקור - Source Browser

כשאתה מתחיל לכתוב יישום, יש לך בוודאי תמונה ברורה למדי של קבצי קוד המקור, המחלקות ופונקציות חברות (Member Functions). אם ברצונך לעבוד על יישום שכתב מישהו אחר, תזדקק לסיוע כלשהו. דפדפן המקור של Visual C++ (או בקיצור, הדפדפן) מאפשר לבחון (וגם לערוך) את היישום מנקודת ראות המחלקה או הפונקציה, ולאו דווקא מנקודת ראות הקובץ. הדפדפן מזכיר במידה מסוימת את כְּלֵי הבחינה ("Inspector") של ספריות מוכוונות-עצמים, כגון Smalltalk. מצבי התצוגה של הדפדפן כוללים:

- ✰ הגדרות והפניות - אתה בוחר רכיב כגון פונקציה, משתנה, סוג, פקודת מאקרו או מחלקה כלשהי, והדפדפן מציג היכן הרכיב מוגדר או מנוצל בפרויקט.
- ✰ קריאה לתרשים / תרשים קורא - לכל פונקציה שתבחר יש ייצוג גרפי לפונקציות שהיא קוראת להן, או אלו שקוראות לה.
- ✰ תרשים מחלקה נגזרת / תרשים מחלקה בסיסית - אלו דיאגרמות היררכיות של מחלקות. למחלקה שנבחרה, הדפדפן מציג מחלקות נגזרות או מחלקות בסיסיות בתוספת איברים. ניתן לשלוט בתצוגת ההרחבה ההיררכית בעזרת העכבר.
- ✰ חלוקת קובץ לרמות - בחר קובץ, והדפדפן יציג את המחלקות, הפונקציות ואיברי הנתונים בצירוף מקומות הגדרתם והשימוש בהם בפרויקט.

✍ **הערה:** אם תארגן מחדש את השורות בקובץ מקור כלשהו, Developer Studio תיצור מחדש את מסד הנתונים של הדפדפן כשתבנה שוב את הפרויקט. כתוצאה מכך, זמן הבנייה מתארך.

בנוסף לדפדפן, Visual C++ כוללת אפשרות לתצוגת מחלקות, ClassView, שאינה תלויה במסד הנתונים של הדפדפן. זוהי תצוגת עץ של כל המחלקות בפרויקט, אשר כוללת פונקציות חברות ואיברי נתונים. לחיצה כפולה על אחד ממרכיבי התצוגה תציג מיידית את קוד המקור שלה. יחד עם זאת, האפשרות ClassView אינה מציגה מידע בסדר ההיררכי שלו, בעוד שהדפדפן עושה זאת.

עזרה מקוונת

בתוכנה Visual C++ מערכת העזרה עברה ליישום נפרד שנקרא תצוגת ספריית MSDN. מערכת העזרה מבוססת על HTML. כל נושא נמצא במסמך HTML נפרד, וכל הדפים משולבים בקבצי אינדקס. תצוגת ספריית MSDN מנצלת קוד שנמצא גם בדפדפן Internet Explorer ולכן היא פועלת בצורה דומה לדפדפן Web שבוודאי מוכר לך. ספריית MSDN מסוגלת לגשת לקבצי עזרה שנמצאים בתקליטור של Visual C++ (זוהי ברירת המחדל של ההתקנה), או בדיסק, וגם לקבצי HTML באינטרנט.

Visual C++ 6.0 מאפשרת להפעיל את העזרה בארבע דרכים:

- ✰ לפי ספר - כאשר תבחר באפשרות Contents (תוכן) בתפריט העזרה של Visual C++, היישום ספריית MSDN עובר למצב תוכן (Content). אפשרות זו מארגנת את התיעוד של Visual Studio, של Visual C++ ושל Win32 SDK בצורה הירכית, לפי ספרים ופרקים.
- ✰ לפי נושא - כאשר תבחר באפשרות Search (חיפוש) בתפריט העזרה של Visual C++, נפתח באופן אוטומטי היישום תצוגת ספריית MSDN. בחר בכרטיסיה Index (אינדקס), ותוכל להקליד מילת מפתח ולהציג את הנושאים והקטעים הקשורים אליה.
- ✰ לפי מילה - כאשר תבחר באפשרות Search (חיפוש) בתפריט העזרה של Visual C++, תופעל ספריית MSDN, עם הכרטיסיה Search פעילה. כאשר כרטיסיה זו פעילה, תוכל להקליד צירוף של מילים ולהציג את הקטעים שמכילים את המילים הללו.
- ✰ מקש העזרה **F1** - זהו ידידו הטוב ביותר של המתכנת. כל שעליך לעשות הוא להציב את הסמן על פקודות הפונקציה, על פקודת המאקרו או על שם המחלקה, ולאחר מכן להקיש על F1 ומערכת העזרה תבצע את כל השאר. אם השם נמצא במספר מקומות (למשל, בקבצי העזרה של MFC ושל Win32), תוכל לבחור את נושא העזרה הרצוי מתוך חלון רשימה.

תהיה הדרך שתבחר לגשת לעזרה המקוונת אשר תהיה, תוכל תמיד להעתיק קטעי טקסט מתוך קבצי העזרה אל התוכנית באמצעות לוח הגזירים.

כלי אבחון של Windows

Visual C++ כוללת מספר כלי אבחון שימושיים (Diagnostic Tools). התוכנית SPY++ מציגה תמונת עץ של התהליכים, של המטלות ושל חלונות המערכת. היא מאפשרת גם להציג הודעות ולבחון את החלונות של היישומים הפועלים ברגע נתון. PVIEW (גרסת Windows9x נקראת PVIEW9x) הוא כלי שימושי לביטול תהליכים שגויים שאינם נראים ברשימת המשימות של Windows 9x (Windows NT Task Manager), מנהל המשימות שמופעל באמצעות לחיצה ימנית בעכבר על גבי סרגל הכלים הוא החלופה של PVIEW). Visual C++ כוללת גם קבוצה שלמה של תוכניות שירות מסוג ActiveX, תוכנית לבדיקת פקדי ActiveX (כרגע ב- Visual C++ עם קוד מקור מלא), The help workshop, מנהל ספריה, תוכניות עיון ועורכי קבצים בינאריים, profiler של קוד מקור ועוד.

בקרת קוד המקור

מיקרוסופט רכשה תוך כדי הפיתוח של Visual C++ 5.0 את הזכויות על מוצר ידוע לבקרת קוד מקור (Source Code Control), בשם SourceSafe. המוצר כלול מאז במהדורה העסקית (Enterprise Edition) של Visual C++ ומשתלב ב- Visual C++ באופן

שמאפשר לתאם פרויקטי תוכנה גדולים. העותק הראשי של קוד המקור של הפרויקט נמצא במקום מרכזי ברשת שהמתכנתים יכולים לגשת אליו ולטעון ממנו עדכוני גרסאות של מודולים. מודולים אלה נשמרים בדרך כלל בכונן הדיסק במחשב האישי של המתכנת. לאחר שהמתכנת מחזיר לרשת מודולים מעודכנים, עמיתיו יכולים לתאם את הגרסאות שבמחשבים שלהם עם העותק הראשי. ניתן לשלב ב- Visual C++גם מערכות אחרות לבקרת קוד מקור.

הגלריה

הגלריה, או Developer Studio Gallery, מאפשרת לשתף רכיבי תוכנה בין פרויקטים שונים. הגלריה מטפלת בשלושה סוגי מודולים:

- ☆ פקדי **ActiveX** - כשמתקינים פקד ActiveX (OCX - בשמו הקודם פקד OLE), נוצרת הגדרה ברישום של Windows (Windows Registry). כל פקדי ActiveX הרשומים מופיעים בחלון הגלריה, ולכן ניתן לבחור אותם לטובת פרויקט כלשהו.
- ☆ מודולי **Visual C++** - כשכותבים מחלקה חדשה, ניתן להוסיף את הקוד שלה לגלריה. מרגע זה ניתן לבחור אותו ולהעתיקו לפרויקטים אחרים. ניתן גם להוסיף משאבים לגלריה.
- ☆ רכיבי **Visual C++** - הגלריה יכולה להכיל כלים שמאפשרים להוסיף רכיבים (Features) לפרויקט קיים, כגון מחלקות, פונקציות, איברי נתונים ומשאבים חדשים. מיקרוסופט מספקת מודולי רכיבים אחדים (למשל, עיבוד זמן סרק, תמיכה בלוח צבעים ומסך Splash) כחלק מחבילת המוצר Visual C++. רכיבים אחרים ניתן יהיה לרכוש מיצרנים אחרים.

טיפ: אם תחליט להשתמש ברכיב מוכלל כלשהו של Visual C++, רצוי לנסותו תחילה עם פרויקט לדוגמה שתיצור במיוחד. אחרת, אם התוצאה שתקבל אינה עומדת בציפיותיך, אתה עלול להתקשות בהסרת הקוד שיתוסף לפרויקט.

ניתן לייבא ולייצא לקבצי OGX את כל רכיבי הגלריה שנוצרים על ידי המשתמש. קבצים אלה הם אמצעים חדשים להפצה ושיתוף של רכיבי Visual C++.

סביבת הפיתוח באמצעות ספריית מחלקת התשתיות - MFC

בחלק זה נציג את גירסה 6.0 של ספריית מחלקת התשתיות של מיקרוסופט (בקיצור, ספריית **MFC**) של סביבת היישום, תוך התמקדות ביתרונותיה. בדוגמה שנביא בהמשך, נציג תוכנית שלדית אך מתפקדת במלואה, אשר תסייע לך להבין את עקרונות סביבת העבודה לפיתוח יישומים. לא נרחיב רבות בהיבט התיאורטי, אך תמצא שהסעיפים העוסקים במיפוי הודעות, מסמכים ותצוגות מכילים מידע חשוב .

סביבת פיתוח יישומים - לשם מה?

אם בכוונתך לפתח יישומים לסביבת Windows, עליך לבחור סביבת פיתוח. בהנחה שסביבות עבודה שאינן תואמות את שפת C, כמו Microsoft Visual Basic ו- Borland Delphi אינן באות בחשבון, לפניך מספר אפשרויות אחרות:

☆ תכנות ב-C באמצעות ממשק תכנות היישומים Win32.

☆ כתיבה עצמית של ספריית מחלקות Windows בשפת ++C, שמבוססת על Win32.

☆ סביבת היישום MFC.

☆ סביבת יישום כלשהי מבוססת Windows, כגון Borland's Object Windows Library (OWL).

אם תתחיל ממש מלא-כלום, כל אפשרות שתבחר תהיה כרוכה בעקומת לימוד תלולה. גם אם יש לך ניסיון כלשהו בתכנות בסביבת Win32 או Win16, עדיין יהיה עליך ללמוד להכיר את ספריית MFC. מהם, אם כן, היתרונות המצדיקים את המאמץ?

אפילו אם הספריה כבר מוכרת לך, כדאי לעבור על רכיביה השונים של בחירה זו.

ספריית **MFC** היא ממשק תכנות היישומים (**API**) של גרסת **Windows** של **++C**. אם תקבל את ההנחה ששפת ++C היא שפת התכנות התקנית לפיתוח יישומים רציניים, אז טבעי של-Windows יהיה ממשק תכנות נפרד משלה. איזה ממשק יהיה טוב יותר מכזה שנוצר על ידי מיקרוסופט, החברה שפיתחה את מערכת ההפעלה Windows? ממשק תכנות זה הוא, למעשה, ספריית MFC.

יישומי סביבת היישום מבוססים על מבנה סטנדרטי. כל מתכנת שמתחיל לעבוד על פרויקט גדול, מפתח תחילה מבנה תוכנית כלשהו. הבעיה היא, שכל מבנה נוצר בהתאם לסגנון האישי של המתכנת, והדבר מקשה על חבר חדש בצוות ללמוד אותו ולפעול לפיו. סביבת היישום של ספריית MFC כוללת מבנה מוגדר של יישום, כזה שהוכיח את עצמו בסביבות תוכנה ובפרויקטים רבים. אם אתה כותב תוכנית לסביבת Windows שמנצלת את ספריית MFC, תוכל לצאת לחופשה בשקט ובשלווה, מתוך ביטחון שהצוות הנושא במשימה יצליח לתחזק ולשפר את התוכנית כדרוש.

אינך צריך לחשוש מכך שהמבנה שמכתיבה ספריית MFC יפגע בגמישות התוכניות שתיצור בעזרתה. ספריית MFC מאפשרת לתוכנית לקרוא לפונקציות Win32 בכל עת, ולכן תוכל לנצל את יתרונות Windows במלואם.

יישומי סביבת היישום קטנים ומהירי ביצוע. בימים ש-Win16 היתה הארכיטקטורה המקובלת, יכולת ליצור קובץ הפעלה מושלם (Self-Contained) בגודל קטן מ- 20KB. כיום, תוכניות של Windows גדולות הרבה יותר. אחת הסיבות לכך היא, שקוד של תוכנית מבוססת 32 סיביות רב יותר. גם במודל הזיכרון הגדול, תוכנית Win16 התבססה על כתובות 16 סיביות עבור משתני מחסנית (Stack Variables) ועבור משתנים גלובליים רבים. תוכניות 32 סיביות מבוססות על כתובות בגודל 32 סיביות בלבד, ולעיתים קרובות מקצות 32 סיביות עבור מספרים שלמים (Integers), מכיון שייצוג כזה יעיל יותר מייצוג של מספרים שלמים ב-16 סיביות בלבד. בנוסף, הקוד החדש לטיפול בחריגים זקוק למשאבי זיכרון גדולים מאוד.

התוכניות הישנות בגודל KB20 היו משוללות סרגל כלים מעוגן (Docking Toolbar), חלונות מפוצלים, אפשרויות הצגה לפני הדפסה, או תמיכה במכולת פקדים (Control Container), רכיבים שכל משתמש מצפה לראותם בתוכניות בימינו. תוכניות MFC גדולות יותר, מכיון שהן עתירות ביצועים ובעלות חזות מצודדת יותר. לשמחתנו, קל היום ליצור יישומים שנקשרים בצורה דינמית לקוד MFC (ולקוד זמן ריצה של C), והדבר מאפשר לצמצם את גודל הקוד מ- 192KB חזרה ל- 20KB בקירוב! הדבר מחייב, כמובן, ספריות DLL גדולות למדי כדי לתמוך בכך, אך אלו מחויבות המציאות.

מבחינת מהירות הביצוע, מדובר בקוד מכונה שנוצר על ידי מהדר בתהליך של אופטימיזציה. הביצוע מהיר, אך ייתכן שתבחין בהשהיה מסוימת בתחילה, שנובעת מטעינת ספריות DLL.

הכלים המובנים של **Visual C++** מצמצמים את תהליך ניפוי השגיאות בתוכנית. עורכי המשאבים של Visual C++, אשף היישום ואשף המחלקה, מצמצמים בצורה משמעותית את הזמן שיש להשקיע בכתיבת הקוד הייחודי ליישום. לדוגמה, עורך המשאבים יוצר קובץ כותר שמכיל ערכים מוגדרים עבור הקבועים במשפטי **#define**. אשף היישום יוצר שלד עבור כל היישום, ואשף המחלקה יוצר תבניות כלליות וגופי פונקציה עבור רכיבי הטיפול בהודעות.

סביבת היישום של ספריית **MFC** הינה עתירת רכיבים. מחלקות ספריית MFC גירסה 1.0 שכלולות בגירסה 7.0 של Microsoft C/C++, כוללות את הרכיבים הבאים:

- ממשק C++ עבור הממשק Windows API.
- מחלקות לשימוש כללי (אינן ייחודיות ל-Windows), שבהן:
  - מחלקות אוסף (Collection) עבור רשימות, מערכים ומפות.
  - מחלקת מחרוזת (String) שימושית ויעילה.
  - מחלקות זמן, משך זמן ותאריך.
  - מחלקות גישה לקבצים ללא תלות בסוג מערכת ההפעלה.
  - תמיכה באחסון ושליפה שיטתיים של עצמים מהדיסק.
- הירכיית מחלקה בסגנון "עצם בעל שורש משותף" (Common Root Object).
- תמיכה ביישומי ממשק רציף רב-מסמכים - MDI (Multiple Document Interface).
- תמיכה כלשהי בגירסה 1.0 של OLE.

גירסה 2.0 של מחלקות ספריית MFC (בגירסה 1.0 של Visual C++) המשיכה את גירסה 1.0 בתמיכה ברכיבים רבים של ממשק משתמש - כאלה שניתן למצוא היום ביישומי Windows - וגם בהנהגת ארכיטקטורת סביבת היישום. להלן רשימת הרכיבים החדשים החשובים ביותר.

- ☆ תמיכה מלאה באפשרויות התפריט Open (פתיחה), Save As (שמירה בשם) ו-Save (שמירה), ובהצגת רשימת הקבצים האחרונים שהיו בעבודה.
- ☆ תמיכה בהצגה לפני הדפסה ובמדפסת.
- ☆ תמיכה בגלילה ובפיצול של חלונות.
- ☆ תמיכה בסרגלי כלים ובשורות מצב (סטטוס).
- ☆ גישה לפקדי Visual Basic של מיקרוסופט.
- ☆ תמיכה בעזרה תלויית-הֶקשר.
- ☆ תמיכה בעיבוד אוטומטי של נתונים שהוקלדו בתיבת דו-שיח.
- ☆ ממשק משופר לגירסה OLE 1.0.
- ☆ תמיכה ב-DLL.

המחלקות בספריה MFC 2.5 (בגירסה Visual C++ 1.0) תרמו את הרכיבים הבאים:

- ☆ תמיכה בקישוריות פתוחה למסדי נתונים - ODBC (Open Database Connectivity) שמאפשרת ליישום הנכתב לגשת אל נתונים במסד נתונים כלשהו ולעדכן אותם. OBDC תומכת במסדי הנתונים המקובלים, כגון Microsoft Access, FoxPro ו-Microsoft SQL Server.
- ☆ ממשק לגירסה 2.01 של OLE שכולל תמיכה בעריכה בתוך הקוד, קישור, גרירה ושחרור והטיפול ב- OLE-Automation.

גירסה 2.0 של Visual C++ היתה גרסת 32 הסיביות הראשונה של המוצר שתמכה בגירסה 3.5 של Windows NT. היא כללה את MFC 3.0, עם הרכיבים הבאים:

- ☆ תמיכה בדו-שיח הכרטיסיות (Tab Dialog, שאינם אלא גליונות תכונות - Property Sheets), שנוספה גם לגירסה 1.51 של Visual C++.
- ☆ סרגלי פקדים צפים שהוטמעו ב-MFC.
- ☆ תמיכה בחלונות בעלי מסגרת צרה.
- ☆ ערכת פיתוח פקדים - CDK (Control Development Kit) נפרדת ליצירת פקדי OLE שמבוססים על 16 או 32 סיביות, למרות שלא ניתנה תמיכה כלשהי במכולת פקדי OLE.

מהדורה מיוחדת של גירסה 2.1 של Visual C++ בשילוב MFC 3.1, הוסיפה את הרכיבים הבאים:

- ☆ תמיכה בפקדים המשותפים של גרסת ביטא החדשה של Windows.
- ☆ דרייבר OBDC (רמה 2) חדש, משולב במנוע מסד הנתונים Access Jet.
- ☆ מחלקות Winsock עבור תקשורת נתונים מסוג TCP/IP.

מיקרוסופט החליטה לדלג על גירסה 3.0 של ++Visual C, והמשיכה ישירות לגירסה 4.0, כדי לסנכרן בין גרסת המוצר לגירסה החדשה של MFC. MFC 4.0 כוללת את הרכיבים הנוספים הבאים:

- ☆ מחלקות של עצמי גישה לנתונים - Data Access Objects) DAO) מבוססי OLE, שנועדו לשימוש יחד עם מנוע Jet.
- ☆ שימוש בסרגלי הפקדים הצפים של Windows 9x במקום אלה של MFC.
- ☆ תמיכה מלאה בפקדים משותפים של Windows 9x, שכוללת תצוגת עץ חדשה ועריכה משופרת של מחלקות תצוגה.
- ☆ מחלקות חדשות לסינכרון מטלות.
- ☆ תמיכה במחלקת פקדי OLE.

גירסה 4.2 של ++Visual C היתה מהדורה חשובה, שכללה את גירסה 4.2 של MFC על הרכיבים החדשים הבאים:

- ☆ מחלקות WinInet.
- ☆ מחלקות שרת ActiveX Documents.
- ☆ מחלקות ActiveX syncronous and syncronous moniker.
- ☆ מחלקות פקדי MFC ActiveX משופרים, ובהן רכיבים כגון הפעלה ללא חלון, קוד ציור שעבר אופטימיזציה וכן הלאה.
- ☆ תמיכה משופרת ב-ODBC, ובכלל זה recordset bulk fetches והעברת נתונים ללא כריכה (Binding).

גירסה 5.0, כוללת את MFC 4.21, שבה יש תיקוני שגיאות תוכנה שהתגלו בגירסה 4.2. לגירסה 5.0 של ++Visual C מספר יתרונות חשובים נוספים:

- ☆ סביבת פיתוח משולבת (IDE) מעוצבת מחדש, Developer Studio 97 שכולל עזרה מקוונת מבוססת HTML ושילוב עם שפות תכנות נוספות, כולל Java.
- ☆ Active Template Library - ATL לבניית פקד ActiveX לאינטרנט.
- ☆ תמיכת שפת ++C עבור תוכניות לקוח מסוג מודל עצם הרכיב - Component) COM Object Model), עם משפט התכנות החדש, import#, עבור ספריות סוג.

המהדורה האחרונה Visual C++ 6.0 כוללת את MFC 6.0 (שים לב שהגרסאות מסונכרנות שוב). הרבה מהרכיבים של MFC 6.0 מאפשרים למפתח לתמוך בפלטפורמה הפעילה החדשה של Microsoft, הכוללת את הדברים הבאים:

- ☆ ספריות MFC שסוגרות את הפקדים המשותפים החדשים של Windows שהוכנסו לראשונה כחלק מ-Internet Explorer 4.0.
- ☆ תמיכה עבור HTML דינמי, שמאפשר למתכנת MFC ליצור יישום שבאופן אוטומטי מפעיל ומארגן דפי HTML .

✰ מכולות מסמכים פעילים, שמאפשרות ליישומים מבוססי MFC להכיל מסמכים פעילים (Active Documents).

✰ תמיכה בתבניות לספקים ולקוחות של OLE DB וחיבורים ל-ADO שעוזרים למפתחי בסיסי נתונים שמשתמשים ב-MFC או ב-ATL.

עקומת הלמידה

כל היתרונות שמנינו נשמעים נהדרים, לא כן? אך כידוע, אין מקבלים דבר חינם. ואמנם, כדי לנצל ביעילות את סביבת היישום יש ללמוד ולהכיר אותה היטב, והדבר נמשך זמן ניכר. אם עליך ללמוד בעת ובעונה אחת את שפת ++C, את Windows ואת ספריית MFC (ללא OLE), יחלפו לפחות 6 חודשים בטרם תתחיל לכתוב תוכנית מעשית אחת. מעניינת העובדה, שלימוד Win32 לבדו נמשך פרק זמן קרוב לזה.

כיצד ייתכן הדבר, אם ספריית MFC כוללת הרבה יותר ממה שכולל Win32? ראשית, תוכל לדלג על פרטים רבים בנושא התכנות, שמתכנתי C בסביבת Win32 נאלצים ללמוד. מנסיוני האישי אוכל להעיד שסביבת יישום מוכוון-עצמים מקלה על תהליך הלימוד של תכנות ב-Windows, בתנאי שהתכנות מוכוון-עצמים (**OOP**) מובן לך.

ספריית MFC לא תהפוך את התכנות המעשי בסביבת Windows לנושא פשוט, בחזקת "תכנות להמונים". מתכנתי יישומים לסביבת Windows משתכרים בדרך כלל טוב יותר ממתכנתים אחרים, ומצב זה יימשך בעתיד הנראה לעין. עקומת הלמידה של ספריית MFC, יחד עם עוצמתה של סביבת היישום, מבטיחים כי מתכנתים שעוסקים בתחום זה ימשיכו להיות מבוקשים בשוק.

מהי מסגרת היישום?

הגדרה אפשרית של מסגרת היישום (Application Framework) היא "אוסף משולב של רכיבי תוכנה מוכווני-עצמים, שכולל את כל הדרוש ליישום כלשהו". זו אינה הגדרה שימושית ביותר, לא כן? אם ברצונך לדעת באמת ובתמים מהי מסגרת היישום, יהיה עליך לקרוא את הספר עד תומו. יחד עם זאת, הדוגמה שנציג בהמשך היא התחלה טובה למדי להכרת הנושא.

מסגרת היישום כנגד ספריית המחלקה

אחת הסיבות ש- ++C היא שפה מקובלת, נעוצה בעובדה שניתן "להרחיב" אותה באמצעות ספריות מחלקה (Class Libraries). מהדרי ++C כוללים מספר ספריות כאלו, יצרנים שונים מציעים ספריות נוספות למכירה, וכמובן שכל בית תוכנה יוצר ספריות כאלו בעצמו לפי הצורך. ספריית מחלקה היא אוסף של מחלקות ++C הקשורות זו לזו, ושניתן לנצל אותן עבור יישום כלשהו. ספריית מחלקה מתמטית, למשל, עשויה לבצע פעולות מתמטיות שכיחות, בעוד שספריית מחלקה של תקשורת תתמוך בוודאי בהעברת נתונים בקישור טורי. לעיתים עליך לבנות עצמים של מחלקה נתונה; לעיתים אתה גוזר מחלקה נפרדת - הכל תלוי בתפיסת התכנון של מחלקת הספריה הנדונה.

סביבת יישום היא קבוצת-על של ספריות מחלקה - אוסף של ספריות. ספריה רגילה היא קבוצה "סגורה" של מחלקות שמיועדות לשילוב בתוכנית מחשב כלשהי, אך סביבת יישום מגדירה את מבנה התוכנית עצמו. מיקרוסופט לא המציאה את רעיון סביבת היישום. הדבר הוצג לראשונה בעולם האקדמי, והגירסה המסחרית הראשונה שלו נועדה למחשב מקינטוש של אפל ונקראה MacApp. מאז יצאה גירסה 2.0 של MFC, ויצרנים נוספים ובהם Borland, הוציאו לשוק מוצרים דומים.

דוגמה של סביבת יישום

עד כה עסקנו בהכללות. הגיעה העת להציג תוכנית, ולא קוד מדומה, כי אם תוכנית אמיתית שניתן להדר ולהריץ באמצעות הספריה MFC. הכרת בוודאי את היישום "Hello, world!", אך הפעם יש בו מספר תוספות (אם עבדת עם גירסה 1.0 של ספריית MFC תכיר את הקוד, להוציא את מחלקת הבסיס של חלון המסגרת). היישום כולל את כמות הקוד הקטנה ביותר הדרושה ליישום Windows שמבוסס על ספריית MFC.

✍ **הערה:** לפי הנוהג שנקבע, שם מחלקה בספריה MFC מתחיל באות C.

לפניך קוד המקור של קבצי הכותר והקוד של היישום MYAPP. שתי המחלקות CMyApp ו-CMyFrame נגזרות ממחלקות הבסיס של ספריית MFC. נתחיל בקובץ הכותר MyApp.h של היישום MYAPP:

```
// application class
class CMyApp : public CWinApp
{
public:
   virtual BOOL InitInstance();
};

// frame window class
class CMyFrame : public CFrameWnd
{
public:
   CMyFrame();
protected:
   // "afx_msg" indicates that the next two functions aree part
   // of the MFC library message dispatch system
   afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
   afx_msg void OnPaint();
                        DECLARE_MESSAGE_MAP()
};
```

ועתה לפניך קובץ הקוד של היישום MYAPP, על פי קובץ MyApp.cpp :

```
#include <afxwin.h>  //MFC library header file declares base
                        classes
#include "myapp.h"

CMyApp theApp; // the one and only CmyApp object

BOOL CMyApp::InitInstance()
{
   m_pMainWnd = new CMyFrame();
   m_pMainWnd->ShowWindow(m_nCmdShow);

   m_pMainWnd->UpdateWindow();
   return TRUE;
}

BEGIN_MESSAGE_MAP(CMyFrame, CFrameWnd)
   ON_WM_LBUTTONDOWN()
   ON_WM_PAINT()
END_MESSAGE_MAP()

CMyFrame::CMyFrame()
{
   Create(NULL, "MYAPP Application");
}

void CMyFrame::OnLButtonDown(UINT nFlags, CPoint point)
{
   TRACE("Entering CMyFrame::OnLButtonDown - %lx, %d, %d\n",
        (long) nFlags, point.x, point.y);
}

void CMyFrame::OnPaint()
{
   CPaintDC dc(this);
   dc.TextOut(0, 0, "Hello, world!");
}
```

נסביר בקצרה אחדים ממרכיבי התוכנית :

הפונקציה **WinMain** - כזכור, Windows מחייבת שהיישום יכיל את הפונקציה WinMain. אינך מבחין בפונקציה זו ביישום, מכיון שהיא מוסתרת בסביבת היישום.

המחלקה **CMyApp** - עצם של המחלקה CMyApp מייצג יישום. התוכנית מגדירה עצם CMyApp גלובלי בודד, theApp. מחלקת הבסיס CWinApp קובעת במידה רבה את הדרך בה העצם theApp פועל.

התחלת פעולת היישום - כשהמשתמש מפעיל את היישום, Windows קוראת לפונקציה WinMain המובנית בסביבת היישום, וזו מחפשת אחר עצם היישום שנבנה לשימוש גלובלי ונמצא במחלקה שנגזרה מ-CWinApp. אל תשכח שבתוכנית ++C, העצמים הגלובליים נוצרים טרם הפעלת התוכנית הראשית (Main).

הפונקציה החברה **CMyApp::InitInstance** - כשהפונקציה WinMain מוצאת את עצם היישום, היא קוראת לפונקציה החברה הווירטואלית InitInstance, וזו מבצעת את הקריאות הדרושות לבנייה והצגת חלון המסגרת הראשי של היישום. עליך "לדרוס" את הפונקציה InitInstance במחלקת היישום הנגזרת, מכיון שמחלקת הבסיס CWinApp אינה יודעת איזה סוג של חלון מסגרת ראשי ברצונך להציג.

הפונקציה החברה **CWinApp::Run** - הפונקציה Run מוסתרת במחלקת הבסיס, אך שולחת את הודעות היישום לחלונות השונים, ועל ידי כך מבטיחה שהיישום ימשיך לפעול. WinMain קוראת ל-Run לאחר שקראה לפונקציה InitInstance.

המחלקה **CMyFrame** - עצם של המחלקה CMyFrame מייצג את חלון המסגרת הראשי של היישום. כשהבנאי (constructor) קורא לפונקציה החברה Create של מחלקת הבסיס CFrameWnd, Windows יוצרת את המבנה בפועל של החלון, וסביבת היישום קושרת אותו אל העצם של ++C. כדי להציג את החלון יש לקרוא גם לפונקציות ShowWindow ו-UpdateWindow, שגם הן פונקציות חברות של מחלקת הבסיס.

הפונקציה **CMyFrame::OnLButtonDown** - זוהי הצגה מוקדמת של אפשרות עיבוד ההודעות של ספריית MFC. בחרנו להצמיד את האירוע "לחצן עכבר שמאלי מטה" לפונקציה החברה CMyFrame. הפונקציה מפעילה את פקודת המאקרו TRACE של ספריית MFC, כדי שזו תציג הודעה בחלון של תוכנית ניפוי השגיאות.

הפונקציה **CMyFrame::OnPaint** - סביבת היישום קוראת לפונקציה חברה חשובה זו, ששייכת למחלקה CMyFrame, בכל פעם שצריך להציג את החלון מחדש : בתחילת התוכנית, כשהמשתמש משנה את גודל החלון וכאשר החלון כולו או חלק ממנו מוצג מחדש. המשפט CPaintDC מתייחס לממשק ההתקן הגרפי - **GDI** (Graphic Device Interface). הפונקציה TextOut מציגה את ההודעה "!Hello, world".

סגירת היישום - המשתמש סוגר את היישום באמצעות סגירת חלון המסגרת הראשי. פעולה זו גורמת לרצף אירועים שמסתיים בפירוק העצם CMyFrame, יציאה מ-Run, יציאה מ-WinMain ופירוק העצם CMyApp.

עיין שוב בדוגמת הקוד, אך הפעם נסה להבחין בתמונה השלמה. רוב מרכיבי התפקוד של היישום נמצאים במחלקות הבסיס CWinApp ו-CFrameWnd של ספריית MFC.

כשיצרנו את היישום MYAPP פעלנו בהתאם למספר כללי מבנה פשוטים וכתבנו פונקציות עיקריות במחלקות הנגזרות. ++C מאפשרת "לשאול" קטעי קוד רבים מבלי להעתיקם בפועל. זוהי מעין שותפות בין המתכנת לסביבת היישום, שסיפקה את המבנה הבסיסי, ואנו הוספנו את הקוד לקבלת יישום ייחודי.

בוודאי התחלת להבין כבר, מדוע סביבת היישום אינה ספריית מחלקה בלבד. לא זאת בלבד שהיא מגדירה את מבנה היישום, היא גם כוללת יותר מאשר מחלקות בסיס של ++C. פגשנו כבר את הפונקציה המוסתרת WinMain וראינו אותה בפעולה. מרכיביה הנוספים של סביבת היישום תומכים בטיפול בהודעות, אבחון תקלות, ספריות DLL ועוד.

מיפוי הודעות על ידי ספריית MFC

הבה נחזור לפונקציה OnLButtonDown שביישום הדוגמה הקודם. חשבת בוודאי שפונקציה זו היא מועמדת אידיאלית לפונקציה וירטואלית. מחלקת בסיס של חלון היתה עשויה להגדיר פונקציות וירטואליות עבור הודעות על אירועי עכבר והודעות רגילות נוספות, ומחלקות חלון נגזרות היו עשויות לבטל אותן בהתאם לצורך. אכן, קיימות ספריות מחלקה של Windows שפועלות בדרך זו.

סביבת היישום של ספריית MFC אינן מנצלות פונקציות וירטואליות עבור הודעות Windows. במקום זאת היא "ממפה" הודעות מוגדרות מסוימות לפונקציות חברות של מחלקה נגזרת בעזרת פקודות מאקרו. מדוע לא להשתמש בפונקציות וירטואליות? נניח ש-MFC הפעילה פונקציה וירטואלית לטיפול בהודעות. המחלקה CWnd יכלה להכריז על פונקציות וירטואליות לטיפול ב-110 הודעות. ++C דורשת טבלת הפצה לפונקציות וירטואליות, **vtable**, לכל מחלקה נגזרת שהתוכנית משתמשת בה. כל טבלת vtable זקוקה להגדרה בגודל 4 בתים לכל פונקציה וירטואלית, ללא קשר אם הפונקציות מתבטלות במחלקה הנגזרת. באופן זה, לכל סוג מוגדר של חלון או פקד, היישום היה זקוק לטבלה בגודל 440 בתים, כדי לתמוך בפונקציות וירטואליות לטיפול בהודעות.

מה בקשר לפונקציות טיפול בהודעות של פקודות תפריט והודעות שמגיעות מאירועי לחיצות בעכבר? אינך יכול להגדיר אותן כפונקציות וירטואליות במחלקת הבסיס של החלון, מכיון שכל יישום עשוי להכיל מערך שונה של פקודות תפריט ולחצנים. מערכת מיפוי ההודעות של ספריית MFC מוותרת על טבלאות vtable גדולות, ומטפלת בהודעות ייחודיות ליישום במקביל לטיפולה בהודעות הרגילות של Windows. מערכת מיפוי ההודעות מאפשרת למספר מחלקות לא-חלונאיות (כגון מחלקות מסמך ומחלקות יישום) לטפל בהודעות פקודה. אין צורך בהרחבות כלשהן לשפת ++C.

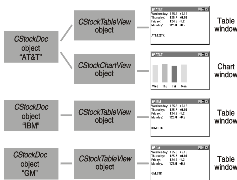
כדי לעבוד במסגרת מפת ההודעות, זקוקה פונקציית MFC המטפלת בהודעה לאב-טיפוס (Prototype), תבנית כללית), גוף והוראות הפעלה (הפעלה על ידי מאקרו). אשף המחלקות יסייע לך להוסיף למחלקותיך פונקציות לטיפול בהודעות. לשם כך עליך לבחור מספר זיהוי של הודעת Windows מתוך תיבת הרשימה, והאשף ייצור את הקוד בצירוף הפרמטרים והערכים המוחזרים.

מסמכים ותצוגות

בדוגמה הקודמת הצגנו עצם יישום (Application Object) ועצם חלון מסגרת (Frame Window Object). רוב יישומי ספריית MFC שתיצור יהיו מורכבים יותר. יישומים כאלה יכללו בדרך כלל מחלקות יישום ומסגרת ושתי מחלקות נוספות שמייצגות את ה"מסמך" וה"תצוגה". הארכיטקטורה מסמך-תצוגה (Document-View) היא הבסיס של סביבת היישום ומבוססת באופן רופף על המחלקות Model/View/Controller של סביבת Smalltalk.

במונחים פשוטים, ארכיטקטורת מסמך-תצוגה (Document-View) מפרידה בין הנתונים והצורה שבה הם מוצגים למשתמש. אחד היתרונות המיידיים של ארכיטקטורה זו הוא האפשרות להציג נתונים זהים בצורות רבות. תאר לעצמך מסמך שמור בדיסק שמכיל את שערי המניות של חודש שלם. המשתמש מעדכן את ערכי המניות באמצעות חלון תצוגת הטבלה, ועל ידי כך משתנה חלון תצוגת התרשים (Chart), מכיון שמקור הנתונים של שני החלונות זהה (למרות שאופן התצוגה שונה).

ביישום ספריית MFC מיוצגים מסמכים ותצוגות על ידי מופעים (Instances) של מחלקות ++C. תרשים 4 מציג שלושה עצמים של המחלקה CStockDoc, שמייצגים שלוש חברות: AT&T, IBM ו-GM. לכל אחד מהמסמכים קשורה תצוגת טבלה, ולאחד מהם יש אפילו תצוגת תרשים. כפי שניתן להבחין, קיימים ארבעה עצמי תצוגה: שלושה עצמים של המחלקה CStockTableView ועצם אחד של המחלקה CStockChartView.



תרשים 4: יחסי הגומלין מסמך-תצוגה

קוד מחלקת הבסיס של המסמך קשור לאפשרויות התפריט File-Open (פתיחת קובץ) ו- File-Save (סגירת קובץ). מחלקת המסמך הנגזרת מבצעת בפועל את הקריאה והכתיבה של נתוני עצם המסמך (סביבת היישום אחראית על רוב פעולות הצגת תיבות הדו-שיח של פתיחת הקובץ וסגירתו, קריאה וכתיבה לקבצים). מחלקת הבסיס של התצוגה מייצגת חלון שנמצא בחלון מסגרת. מחלקת התצוגה הנגזרת פועלת הדדית עם מחלקת המסמך הקשורה אליה, ומטפלת בתצוגת המסמך ובתקשורת עם המדפסת. מחלקת התצוגה הנגזרת ומחלקות הבסיס שלה מטפלות בהודעות Windows. הספריה MFC שולטת בכל יחסי הגומלין המתקיימים בין מסמכים, תצוגות, חלונות מסגרת ועצם היישום, על פי רוב באמצעות פונקציות וירטואליות.

עצם מסמך אינו קשור בהכרח לקובץ בדיסק שנטען בשלמותו לזיכרון. אם לדוגמה "מסמך" היה באמת מסד נתונים, היה ניתן לדרוס פונקציות-חברות נבחרות של מחלקת המסמך, ואפשרות התפריט File-Open היתה מציגה רשימת מסדי נתונים במקום רשימת קבצים.

צעדים ראשונים עם אשף היישומים - "!Hello, World"

בחלק הקודם למדנו על ארכיטקטורת מסמך-תצוגה (Document-View) בגירסה 6.0 של ספריית MFC. בחלק זה נלמד ליצור יישום פונקציונלי של ספריית MFC, גם אם לא ניכנס לעומק המורכבות של היררכיית המחלקות ויחסי הגומלין בין עצמים. נעבוד עם מרכיב אחד בלבד של תוכנית מסמך-תצוגה, אשר קשור בצורה הדוקה לחלון. לפי שעה נוכל להתעלם ממרכיבים כגון מחלקת היישום, חלון המסגרת והמסמך. מובן שהיישום שניצור לא יוכל לשמור את נתוניו בדיסק, וגם לא יתמוך בתצוגות רבות, אך כל זה אפשרי כמובן לבצע עם MFC.

מפאת החשיבות הרבה של משאבים ביישום מבוסס Windows, ננצל את ResourceView כדי לחקור חזותית את משאבי התוכנית החדשה שנכתוב. גם נציג מספר רמזים באשר להגדרת סביבת Windows באופן שישפר את מהירות בניית היישום ויאפשר אופטימיזציה של פלט ניפוי השגיאות.

מהי תצוגה?

מנקודת ראות המשתמש, תצוגה (View) היא חלון רגיל שניתן לשנות את גודלו, להזיז או לסגור אותו, כפי שניתן לעשות בכל יישום מבוסס Windows. מנקודת מבט המתכנת, תצוגה היא עצם ++C במחלקה שנגזרה מהמחלקה CView של ספריית MFC. בדומה לעצם ++C כלשהו, התנהגות עצם התצוגה נקבעת על ידי פונקציות-חברות (ואיברי נתונים) של המחלקה - הן הפונקציות המיוחדות ליישום במחלקה הנגזרת, והן הפונקציות הסטנדרטיות שעברו בירושה מהמחלקות הבסיסיות.

ב- ++Visual C ניתן ליצור יישומים מעניינים לסביבת Windows על ידי הוספת קוד למחלקת התצוגה הנגזרת שיוצר מחולל הקוד של אשף היישומים. כשמריצים את התוכנית, סביבת היישום של ספריית MFC בונה עצם של מחלקת התצוגה הנגזרת ומציגה חלון שקשור באופן הדוק לעצם התצוגה של ++C. כפי שמקובל בתכנות ב- ++C, קוד מחלקת התצוגה מתחלק לשני מודולי מקור: קובץ הכותר (H.*) וקובץ היישום (CPP.*).

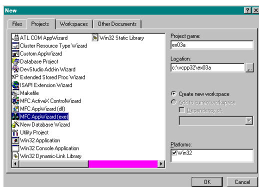
ממשק מסמך בודד כנגד ממשק מרובה מסמכים

ספריית MFC תומכת בשני סוגי יישומים שונים זה מזה: ממשק מסמך בודד - **SDI** (Single Document Interface) וממשק מרובה מסמכים - **MDI** (Multiple Document Interface). מבחינת המשתמש, ליישום SDI יש חלון אחד בלבד. אם היישום תלוי ב"מסמכים" שנמצאים בקבצי דיסק, ניתן לטעון מסמך אחד בלבד בכל זמן נתון. היישום NotePad (פנקס רשימות) המקורי של Windows הוא דוגמה ליישום SDI. ליישום MDI יש חלונות-בנים (Child Windows) אחדים, שכל אחד מהם שייך למסמך נפרד. מעבד התמלילים Word של מיקרוסופט הוא דוגמה טובה ליישום MDI.

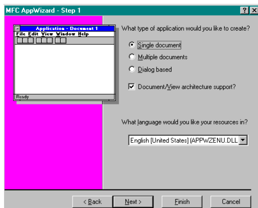
היישום "שאינו עושה דבר" - EX03A

אשף היישומים יוצר קוד של יישום פונקציונלי מספריית MFC. פעולתו של יישום זה מתמצית בהצגת חלון ריק ולצידו תפריט. בהמשך תוסיף ליישום קוד שמצייר בתוך החלון. בנה את היישום בהתאם לשלבים הבאים:

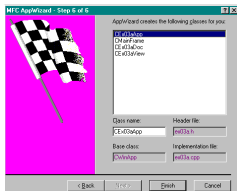
1. הרץ את אשף היישומים וצור בעזרתו קוד מקור של יישום **SDI**. בחר באפשרות **New** (חדש) מתפריט **File** (קובץ) של ++Visual C, ולאחר מכן לחץ על הכרטיסיה **Projects** בתיבת הדו-שיח **New** שנפתחה, ראה דוגמה:



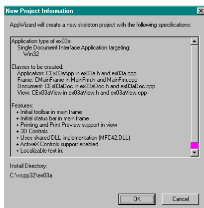
ודא שסימנת את האפשרות **MFC AppWizard (exe)**, ולאחר מכן הקלד **C:\vcpp32** (או שם תיקיה אחר שבחרת) בתיבת העריכה **Location**. הקלד **ex03a** בתיבת העריכה **Project Name**, ולאחר מכן לחץ על לחצן OK. כעת התקדם ברצף המסכים של AppWizard, שהראשון שבהם מוצג לפניך:



ודא שבחרת באפשרות **Single Document**. אשר את הגדרות המחדל בארבעת המסכים הבאים. המסך האחרון ייראה כך:



שים לב ששמות המחלקות ושמות קבצי המקור מבוססים על שם הפרויקט EX03A. תוכל לשנות את השמות בשלב זה, אם רצונך בכך. לחץ על לחצן **Finish**. רגע לפני שאשף היישומים יתחיל ליצור את הקוד תוצג תיבת הדו-שיח New Project Information, שנראית כך:



כאשר תלחץ על OK לאישור, אשף היישומים יתחיל ליצור תת-ספריה עבור היישום החדש (ex03a) וימלא אותה בסדרת קבצים. בסיום פעולתו, בדוק את תת-הספריה של היישום. הקבצים הבאים מעניינים אותנו בשלב זה:

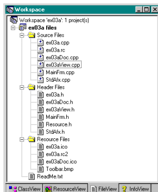
קובץ	תיאור
ex03a.dsp	קובץ פרויקט שמאפשר ל- Visual C++ לבנות את היישום.
ex03a.dsw	קובץ שטח עבודה שמכיל הגדרה יחידה עבור ex03a.dsp.
ex03a.rc	קובץ ASCII שמכיל תסריט משאבים.
ex03aView.cpp	קובץ יישום מחלקת תצוגה שמכיל פונקציות-חברות של המחלקה CEx03aView.
ex03aView.h	קובץ כותר של מחלקת תצוגה, שמכיל הכרזה על המחלקה CEx03aView.
ex03a.opt	קובץ בינארי שאומר ל- Visual C++ איזה קבצים פתוחים עבור הפרויקט, וכיצד מסודרים החלונות (קובץ זה לא נוצר עד שאתה שומר את הפרויקט).
ReadMe.txt	קובץ טקסט שמסביר את ייעוד הקבצים שנוצרו.
resource.h	קובץ כותר המכיל הגדרות #define של קבועים (Constants).

פתח את הקבצים ex03aView.cpp ו- ex03aView.h ועיין בקוד המקור. קבצים אלה מגדירים יחד את המחלקה CEx03aView, שלה תפקיד מרכזי ביישום. עצם של המחלקה CEx03aView מתאים לחלון התצוגה של היישום שבו תתרחש רוב הפעילות.

2. הדר וקשר את הקוד שנוצר. נוסף ליצירת קוד, אשף היישומים יוצר גם קובץ פרויקט מותאם וקובץ שטח עבודה עבור היישום החדש. קובץ הפרויקט, ex03a.dsp, מציין את כל הרכיבים התלויים (dependencies) יחד עם דגלי האפשרויות של תהליכי ההידור והקישור. הפרויקט החדש הופך לפרויקט הנוכחי של Visual C++, ולכן תוכל ליצור עתה את היישום באמצעות בחירת האפשרות Build Ex03a מתוך התפריט Build, או באמצעות לחיצה על לחצן סרגל הכלים הבא:

אם לא יהיו תקלות כלשהן בתהליך הבנייה, ייווצר קובץ ההפעלה ex03a.exe בתת-ספריה חדשה, Debug, תחת ex03a\. קבצי OBJ וקבצי ביניים נוספים נשמרים אף הם בספריה Debug. השווה את מבנה הספריה בדיסק עם זה שבדף FileView של החלון Workspace.

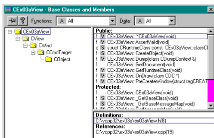
הדף FileView מכיל תצוגה לוגית של הפרויקט. קבצי הכותר מוצגים תחת Header Files, למרות שהם נמצאים באותה תת-ספריה של קבצי CPP. קבצי המשאבים נמצאים בתת-הספריה res\.



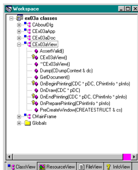
3. בדוק את היישום שנוצר. בחר באפשרות **Execute Ex03a.exe** מתוך התפריט **Build**. נסה להריץ את התוכנית. ביצועיה דלים למדי, לא כן? (גם הקוד שלה אינו רב). למעשה, שיערת בוודאי שהתוכנית מכילה רכיבים רבים, אך עדיין לא הפעלנו את כולם. בסיום נסיונותיך, סגור את חלון התוכנית.

4. דפדף ביישום. בחר באפשרות **Source Browser** מתפריט **Tools**. אם לא הוגדרה בפרויקט יצירת מסד נתונים, ++Visual C תציע לך לשנות את ההגדרות ותהדר עבורך את התוכנית. אם ברצונך לשנות את ההגדרות בעצמך, בחר באפשרות **Settings** מתפריט **Project**. בדף ++C/C לחץ על האפשרות Generate Browse Info. בדף Browse Info, לחץ על Build Browse Info File.

כשיופיע החלון Browse, בחר את Base Classes And Members, ולאחר מכן הקלד CEx03aView. לאחר שתרחיב את התצוגה ההיררכית יופיע מסך כגון זה:



השווה את תצוגת העיון לזו של ClassView בחלון Workspace:



חלון ClassView אינו מציג את היררכיית המחלקות, אך באותה מידה גם אינו כולל את התקורה הנוספת של תצוגת העיון. אם אתה מסתפק בתצוגה של ClassView, אל תטרח לבנות את מסד הנתונים של תצוגת העיון.

מחלקת התצוגה CEx03aView

אשף היישומים יצר את מחלקת התצוגה CEx03aView, שהינה ייחודית ליישום EX03A (אשף היישומים יוצר מחלקות על פי שם הפרויקט שצוין בתיבת הדו-שיח הראשונה שלו). המחלקה CEx03aView נמצאת בסוף שרשרת ההורשה של מחלקות הספריה MFC, כפי שראינו קודם לכן בחלון Browse. במהלך ההורשה, המחלקה אוספת פונקציות-חברות ואיברי נתונים. תוכל להמשיך וללמוד אודות המחלקות הללו בספר Microsoft Foundation Class Reference, אך עליך להביט בכל תיאורי המחלקות הבסיסיות, מכיון שתיאורי הפונקציות-החברות העוברות בירושה אינם חוזרים בדרך כלל במחלקות הנגזרות.

המחלקות הבסיסיות החשובות ביותר של CEx03aView הן CWnd ו-CView. המחלקה CWnd מספקת את המאפיינים החלונאיים של CEx03aView, ואילו CView מספקת את הקשרים לשאר סביבת היישום, ובמיוחד למסמך ולחלון המסגרת.

כתיבה בחלון התצוגה -
ממשק ההתקן הגרפי של GDI - Windows

הכל מוכן עתה ליצירת קוד שיכתוב בחלון התצוגה. בשלב זה נערוך מספר שינויים ישירות בקוד המקור של EX03A.

הפונקציה-החברה OnDraw

OnDraw היא פונקציה-חברה וירטואלית של המחלקה CView, שנקראת על ידי סביבת היישום בכל פעם שיש צורך להציג מחדש את חלון התצוגה. הצורך להציג את החלון מחדש מתעורר בכל פעם שהמשתמש משנה את גודל החלון, חושף חלק שהיה מוסתר עד כה, או שהנתונים שמוצגים בו משתנים. אם המשתמש משנה את גודל החלון או חושף חלק שהיה מוסתר עד כה, סביבת היישום קוראת לפונקציה OnDraw; אך אם פונקציה כלשהי בתוכנית משנה את הנתונים, עליה להודיע ל-Windows אודות השינוי באמצעות קריאה לפונקציה-החברה היורשת (inherited) של התצוגה, Invalidate (או InvalidateRect). קריאה זו ל-Invalidate מפעילה בהמשך קריאה לפונקציה OnDraw.

למרות שניתן לכתוב בחלון בכל עת, מומלץ לאפשר ל-Windows לצבור את השינויים ולטפל בהם בפעם אחת באמצעות הפונקציה OnDraw. באופן זה התוכנית תוכל להגיב לאירועים שנוצרו על ידיה ולאירועים שנוצרו על ידי מערכת ההפעלה, כגון שינויי גודל.

הֶקשֵר ההתקן של Windows

Windows אינה מאפשרת גישה ישירה לחומרה של מסך, אלא מתקשרת איתה באמצעות יישות שנקראת "הֶקשֵר ההתקן" (device context) הקשורה לחלון. הקשר ההתקן בספריה MFC הוא למעשה עצם ++C של המחלקה CDC, אשר מועבר (באמצעות

מצביע, pointer) כפרמטר אל הפונקציה OnDraw. לאחר שנמצא בידך המצביע של הקשר ההתקן, תוכל לקרוא לפונקציות-החברות הרבות של CDC והן תבצענה את פעולת הכתיבה.

הוספת קוד כתיבה לתוכנית EX03A

כעת הבה נכתוב את הקוד לכתיבת טקסט כלשהו ולציור מעגל בחלון התצוגה. פתח את הפרויקט EX03A ב-Visual C++ אם לא עשית זאת עד עתה. תוכל לנצל את ClassView של חלון Workspace לאיתור קוד הפונקציה (באמצעות לחיצה כפולה על OnDraw), או לפתוח את קובץ קוד המקור ex03aView.cpp מתוך FileView ולאתר את הפונקציה בעצמך.

1. ערוך את הפונקציה **OnDraw** בקובץ **ex03aView**. מצא את קוד הפונקציה שנוצר על ידי אשף היישומים. כך הוא נראה:

```
void CEx03aView::OnDraw(CDC* pDC)
{
    CEx03aDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

   // TODO: add draw code for native data here
}
```

קטע הקוד הבא שמופיע בהצללה (זה שהקלדת), מחליף את הקוד הקודם:

```
void CEx03aView::OnDraw(CDC* pDC)
{
pDC->TextOut(0, 0, "Hello, world!"); // prints in default font
                                    // & size, top left corner
  pDC->SelectStockObject(GRAY_BRUSH);   // selects a brush for
                                         // the circle interior
  pDC->Ellipse(CRect(0, 20, 100, 120));// draws a gray circle
                                        // 100 units in diameter
}
```

תוכל לבטל את הקריאה ל-GetDocument ללא חשש, מכיון שעדיין אין אנו מטפלים במסמך. הפונקציות TextOut, SelectStockObject ו-Ellipse הן כולן פונקציות-חברות של מחלקת הקשר ההתקן, CDC, השייכת לסביבת היישום. הפונקציה Ellipse מציירת מעגל אם אורך המלבן המגביל שווה לרוחבו.

הספריה MFC כוללת מחלקה שימושית בשם CRect, שמיועדת למלבני Windows. עצם CRect זמני משמש כארגומנט של המלבן המגביל עבור הפונקציה שמציירת את האליפסה. נפגוש את הפונקציה CRect בדוגמאות נוספות בהמשך.

2. הדר את **EX03A** פעם נוספת והפעל אותו לניסיון. בחר באפשרות **Build** מתוך תפריט **Project**. אם ההידור עבר ללא תקלות כלשהן, נסה להפעיל את היישום פעם נוספת. כעת יש לך תוכנית שסוף-סוף מפגינה ביצועים!

למתכנתי Win32

הפונקציה WinMain ופונקציות הפרוצדורה של Windows מוסתרות בסביבת היישום. נפגוש בהן בהמשך, כאשר נלמד על מסגרת ספריית MFC ומחלקות היישום. לפי שעה, בוודאי שאלת את עצמך מה קרה להודעה WM_PAINT. ציפית בוודאי לשרטט בחלון בתגובה להודעה זו, וגם ציפית לקבל את ידית הקשר ההתקן ממבנה PAINTSTRUCT שמחזירה הפונקציה BeginPaint של Windows.

סביבת היישום ביצעה עבורך את כל העבודה "השחורה" ושרתה כהקשר התקן (בצורת מצביע עצם) בפונקציה הווירטואלית OnDraw. פונקציות וירטואליות אמיתיות במחלקות Windows הן תופעה נדירה בספריית MFC. פונקציות מיפוי ההודעות של הספריה מופצות על ידי סביבת היישום ומטפלות ברוב הודעות מערכת ההפעלה. מתכנתים שעבדו עם גירסה 1.0 של MFC, נאלצו להגדיר פונקציית מיפוי הודעה OnPaint עבור מחלקות החלון הנגזרות שלהם. מגירסה 2.5 ואילך לעומת זאת, OnPaint מופתה במחלקה CView ויצרה קריאות רב צורתיות (פולימורפיות) לפונקציה OnDraw. הסיבה לכך: OnDraw צריכה לתמוך גם במדפסת. הפונקציות OnPaint ו-OnPrint קוראות ל-OnDraw ועל ידי כך מאפשרות לכתוב למדפסת ולצג באמצעות קוד זהה.

הצגה מוקדמת של עורכי המשאבים

כעת, כשבידך תוכנית יישום מושלמת, זו ההזדמנות לסקירה מהירה על עורכי המשאבים. למרות שתסריט המשאבים ex03a.rc של היישום הוא קובץ ASCII, לא מומלץ לעדכן אותו בעזרת עורך טקסט פשוט. עורך המשאבים (resource editor) הוא הכלי המיועד לכך.

תוכן תסריט קובץ המשאבים ex03a.rc

קובץ המשאבים קובע במידה רבה את החזות ודרך הפעולה של היישום EX03A. הקובץ מכיל את (או מצביע אל) משאבי מערכת ההפעלה הבאים:

משאב	תיאור
Accelerator (מאיץ)	הגדרות מקשים והדמיית אפשרויות תפריט וסרגל הכלים.
Dialog (תיבת דו-שיח)	עיצוב ותכנים של תיבות דו-שיח. EX03A כולל רק את תיבת דו-שיח About.
Icon (סמל)	סמלים (גרסאות בגודל של 16x16 או 32x32 פיקסלים), כגון סמלי היישומים של הסייר של Windows ואחרים. EX03A מנצל את הלוגו של MFC כסמל ההפעלה של היישום.
Menu (תפריט)	התפריט הראשי של היישום והתפריטים המוקפצים (pop-up) הקשורים אליו.

משאב	תיאור
String table (טבלת מחרוזות)	מחרוזות שאינן מהוות חלק מקוד המקור של ++C.
Toolbar (סרגל כלים)	טור הלחצנים שנמצא מתחת לתפריט.
Version (גירסה)	תיאור התוכנית, מספר הגירסה, שפה וכן הלאה.

נוסף למשאבים שתוארו, הקובץ ex03a.rc מכיל את משפטי התכנות הבאים:

```
#include "afxres.h"
#include "afxres.rc"
```

תפקיד משפטים אלה לספק ליישום גישה למשאבי MFC מסוימים, שכוללים מחרוזות, לחצנים גרפיים ומרכיבים הדרושים להדפסה ולקישור והטבעת עצמים (OLE).

✍ **הערה:** אם אתה עובד עם גרסת DLL המשותפת של ספריית MFC, המשאבים המשותפים שמורים בספריית DLL של MFC.

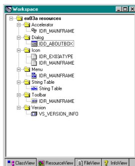
הקובץ ex03a.rc מכיל גם את המשפט הזה:

```
#include "resource.h"
```

משפט זה משלב ביישום שלושה קבועי define#: IDR_MAINFRAME (שמזהה את התפריט, הסמל, רשימת המחרוזות וטבלת המאיצים); IDR_EX03ATYPE (שמזהה את סמל המחדל של היישום, אשר אינו בשימוש בתוכנית שלנו); ו-IDD_ABOUTBOX (שמזהה את תיבת הדו-שיח 'על אודות'). קובץ כותר זה נכלל בתוכנית באופן עקיף על ידי קבצי המקור של היישום. אם תשלב בתוכנית קבועים נוספים (סמלים), ההגדרות תופענה בסופו של דבר בקובץ resource.h. אם אתה עורך קובץ זה במצב טקסט, עשה זאת בזהירות, מכיון שהשינויים שהכנסת עלולים להתבטל בפעם הבאה שתפתח אותו בעזרת עורך משאבים.

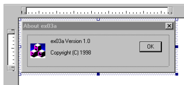
הפעלת עורך משאב תיבת דו-שיח

1. פתח את קובץ המשאבים (RC) של הפרויקט. לחץ על לחצן **ResourceView** שבחלון **Workplace**. אם תרחיב כל אחד מהפריטים המוצגים, תתגלה לפניך התמונה הבאה בחלון העורך.



2. בחן את משאבי היישום. כעת הקדש את הזמן הדרוש והבט בעיון על המשאבים עצמם. כשבוחרים משאב כלשהו באמצעות לחיצה כפולה, נפתח חלון נוסף ובו הכלים המתאימים למשאב שנבחר. אם תפתח משאב תיבת דו-שיח (Dialog Resource), למשל, יופיע פקד תיבת כלים. אם לא מופיע הכלי הדרוש, לחץ לחיצה ימנית בסרגל כלים כלשהו ולאחר מכן סמן את האפשרות Controls (פקדים).

3. שנה את תיבת הדו-שיח **IDD_ABOUTBOX**. ערוך שינויים בתיבת הדו-שיח About Ex03a המוצגת לפניך:



תוכל לשנות את גודל החלון באמצעות גרירת הגבול הימני או התחתון שלו, להזיז את לחצן OK, לשנות את הטקסט וכו׳. כל שעליך לעשות הוא ללחוץ על מרכיב כלשהו, כדי לבחור אותו, ואחר כך ללחוץ לחיצה ימנית בעכבר כדי לשנות את מאפייניו.

4. בנה את הפרויקט מחדש בעזרת קובץ המשאבים המעודכן. ב- Visual C++, בחר באפשרות **Build Ex03a.exe** מתוך התפריט **Build**. שים לב, אין צורך בהידור חוזר; Visual C++ שומרת את קובץ המשאבים המעודכן, ולאחר מכן מהדר המשאבים (rc.exe) מהדר את קובץ ex03a.rc. הקובץ ex03a.res שנוצר נשלח אל תוכנית הקישור (linker). זו האחרונה פועלת במהירות, מכיוון שביכולתה לקשר את הפרויקט בצורה מדורגת.

5. בדוק את גרסת היישום החדשה. הרץ פעם נוספת את התוכנית EX03A ובחר באפשרות **About** מתוך תפריט העזרה של היישום, כדי לוודא שהשינויים שערכת אכן מופיעים.

Win32 Debug Target בהשוואה ל- Win32 Release Target

אם תפתח את תיבת הרשימה של סרגל הכלים Build, תוכל לראות בו שני פריטים: Win32 Debug Target ו- Win32 Release Target (סרגל כלים זה אינו מופיע מעצמו, ויש להציגו על ידי בחירת האפשרות Customize בתפריט Tools). פריטים אלה הם מטרות (Targets) שמייצגות קבוצות מוגדרות של אפשרויות בנייה. כשאתה היישומים יוצר פרויקט, הוא יוצר שתי מטרות מחדל שנבדלות בהגדרותיהן, כפי שמוצג בטבלה שלהלן:

טבלה 2: הצגת שתי מטרות מחדל הנבדלות בהגדרותיהן

	בנייה בשיטת **Release**	בנייה בשיטת **Debug**
ניפוי שגיאות בקוד המקור	לא מאופשר	אפשרי באמצעות המהדר ותוכנית הקישור כאחד
פקודות מאקרו לאבחון	לא מאופשרות (NDEBUG מוגדר)	מאופשר (DEBUG_ מוגדר)
זיקה לספריה	ספריית MFC Release	ספריית MFC Debug
אופטימיזציה של המהדר	אופטימיזציה של מהירות (לא נמצאת בגרסת הלימוד)	ללא אופטימיזציה (הידור מהיר)

מפתחים את היישום במצב Debug ולאחר מכן בונים אותו מחדש במצב Release לפני מסירתו למשתמש בגרסת זמן ריצה. קובץ ההפעלה שנבנה במצב Release קטן ומהיר יותר בביצוע, בהנחה שניפית את כל שגיאות התכנות. בוחרים את התצורה מתוך חלון המטרה של הבנייה, כמתואר בתרשים 2. כברירת מחדל, קבצי הפלט של ניפוי השגיאות וקבצי הביניים שנוצרים בתהליך, נשמרים בתת-הספריה Debug; קבצי Release נשמרים בתת-הספריה Release. תוכל לשנות את נתיב שמירת הקבצים באמצעות הכרטיסיה General שבתיבת הדו-שיח Project Setting.

תוכל לערוך תצורה אישית מותאמת באמצעות בחירת האפשרות Configurations מתפריט Build של ++Visual C.

הפעלת פקודות המאקרו לאבחון שגיאות

פקודות המאקרו TRACE של סביבת היישום שימושיות במיוחד למעקב אחר פעילות התוכנית (דיאגנוסטיקה, אבחון שגיאות). הן מחייבות להפעיל את רכיב העקיבה (Tracing), וזו גם אפשרות המחדל. אם אינך מבחין בפלט TRACE מהתוכנית, ודא

תחילה שאתה מריץ אותה במצב ניפוי (Debug) ולאחר מכן הרץ את תוכנית השירות TRACER. אם תסמן את תיבת הסימון Enable Tracing, תראה ש-TRACER תכניס את המשפט

```
TraceEnabled = 1
```

בקטע [Diagnostics] בקובץ Afx.ini שבספריה Windows (אין זה מערך הרישום Registry) תוכל לנצל את TRACER כדי לקבל פלטי אבחון של MFC, וגם כדי לשלב מידע מתוך מערך ההודעות, OLE, מסד נתונים והאינטרנט.

הבנת הקבצים המהודרים מראש

כאשר אשף היישומים מחולל את היישום, הוא מארגן הגדרות מתג וקבצים עבור כותרים מהודרים מראש (Precompiled Headers). אתה חייב להבין באיזו שיטה מערכת הבנייה של היישום מטפלת בכותרים מהודרים מראש, כדי שתוכל לנהל את הפרויקטים שלך בצורה יעילה.

הערה: לשפת Visual C++ יש שתי "מערכות" כותרים מהודרים מראש: מערכת אוטומטית ומערכת ידנית. כותרים מהודרים אוטומטיים מופעלים באמצעות מתג ההידור Yx/, ושומרים את הפלט שמפיק המהדר בקובץ דמוי מסד נתונים. כותרים מהודרים ידניים מופעלים באמצעות המתגים Yc/ ו- Yu/, ותופסים מקום מרכזי בכל הפרויקטים שנוצרים על ידי אשף היישומים.

כותרים מהודרים מראש מייצגים "תמונות בזק" שצולמו לאחר שורה מסוימת של קוד המקור. בתוכניות של ספריית MFC, תמונת הבזק מצולמת בדרך כלל מייד לאחר המשפט הבא:

```
#include "StdAfx.h"
```

קובץ StdAfx.h מכיל משפטי include# שמיועדים לקבצי הכותר של ספריית MFC. תוכן הקובץ תלוי באפשרויות שבחרת בעת הפעלת אשף היישומים, אך תמיד יכלול את המשפטים הבאים:

```
#include <afxwin.h>
#include <afxext.h>
```

אם אתה עובד עם מסמכים מורכבים, קובץ StdAfx.h יכיל גם את ההגדרה הבאה:

```
#include <afxole.h>
```

ואם אתה עובד עם Automation או עם פקדי ActiveX, הקובץ יכיל גם את ההגדרה:

```
#include <afxdisp.h>
```

אם אתה עובד עם הפקדים המשותפים של Internet Explorer 4.0, הקובץ יכיל גם את ההגדרה:

```
#include <afxdtctl.h>
```

מפעם לפעם תזדקק לקבצי כותר אחרים, כמו למשל הכותר הדרוש למחלקות אוסף מבוססות-תבנית (template-based collection classes), שפונים אליו באמצעות המשפט

```
#include <afxtempl.h>
```

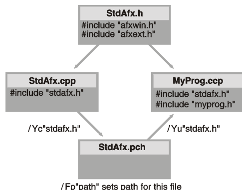
קובץ המקור StdAfx.cpp מכיל הגדרה אחת בלבד:

```
#include "StdAfx.h"
```

הוא משמש ליצירת קובץ כותר מהודר מראש בספריית הפרויקט. כותרי הספריה של MFC הכלולים בקובץ הכותר StdAfx.h אינם משתנים לעולם, אך תהליך ההידור שלהם נמשך זמן רב. מתג המהדר /Yc שמיועד עבור StdAfx.cpp בלבד, גורם ליצירת קובץ כותר מהודר מראש - **PCH** (Precompiled Header). המתג /Yu שמיועד לכל שאר קבצי קוד המקור, גורם לשימוש בקובץ PCH שקיים. המתג /Fp מציין את שם קובץ PCH שהיה מקבל את שם הפרויקט (בתוספת הסיומת PCH) בספריית פלט המטרה. תרשים 5 מתאר את התהליך בשלמותו.

אשף היישומים משלב עבורך את המתגים /Yc ו- /Yu, אך תוכל לשנות זאת אם ברצונך בכך. ניתן להתאים הגדרות שונות של מתגי המהדר לקבצי מקור שונים. בכרטיסיה C/C++ שבתיבת הדו-שיח Project Settings, תראה את המתג /Yc כאשר תבחר את StdAfx.cpp בלבד. הגדרה זו מבטלת את /Yu שהוגדר עבור המטרה.

הֱיה ער לעובדה שקבצי PCH הם גדולים, 5MB הוא גודל טיפוסי. אם לא תנהג בזהירות, תמלא את הדיסק עד אפס מקום. תוכל לשלוט במצב על ידי מחיקה תקופתית של ספריות Debug של הפרויקט, או לנצל את אפשרות המהדר /Fp כדי לנתב קבצי PCH לספריה משותפת.



תרשים 5: תהליך יצירת קובץ כותר מהודר מראש של Visual C++

שתי דרכים להרצת תוכנית

התוכנה Visual C++ מאפשרת להריץ את התוכנית באופן ישיר (בעזרת הקשה על Ctrl+F5), או באמצעות תוכנית ניפוי השגיאות (בהקשה על F5). הרצה ישירה מהירה יותר, מכיון ש- Visual C++ אינה צריכה לטעון את תוכנת הניפוי (Debugger) לפני כן. אם אינך רוצה בהודעות האבחון, או אינך רוצה להציב נקודות עצירה בתוכנית (Breakpoints), הפעל את התוכנית באמצעות צירוף המקשים Ctrl+F5, או לחץ על לחצן "סימן הקריאה" שבסרגל הכלים Build.

עד פה היתה ההיכרות עם MFC. אם בחרת להמשיך בלימוד MFC ונושאים מתקדמים אחרים בתכנות בסביבת Windows, הוצאת הוד-עמי מציעה לך שני ספרים:

- Visual C++ 6 סדנת לימוד
- המדריך השלם Visual C++ 6

אתה בוודאי שואל מה ההבדלים בין השניים; ובכן, שניהם עוסקים ב-MFC. אם הספר "Visual C++ 6 סדנת לימוד" ילמד אותך נהיגה, אז הספר "המדריך השלם לשפת Visual C++ 6" ילמד אותך גם נהיגה וגם מכונאות רכב.

לשני הספרים מצורף תקליטור ובו כל קוד המקור של הדוגמאות שבספר.

לשני הספרים אינדקס ענק באנגלית בו תוכל למצוא כל מושג, מונח, פונקציה או מחלקה.

תוכל לעיין בתוכן העניינים המלא של ספרים אלה ופרק לדוגמה דרך הקטלוג הנמצא בתקליטור המצורף לספר זה, או באתר האינטרנט של הוצאת הוד-עמי בכתובת http://www.hod-ami.co.il.

Index אינדקס

D

E

F

G

H

I

K

L

M

R

S

הוצאת הוד-עמי ספרי מחשבים
ת.ד. 6108 הרצליה 46160 **הזמנות באתר**
info@hod-ami.co.il www.**hod-ami**.co.il

קטלוג ינואר 2017

מחיר*	עמ'	כולל	
			אינטרנט - מפתחי אתרים/גרפיקה
29	256		אמא, אבא - בניתי אתר באינטרנט (HTML)
249	300		הגדל את הכנסות העסק שלך באמצעות **פרסום בגוגל Google AdWords**
159	390		**HTML5** המדריך לבניית אתרים ולמערכות WEB, הדור הבא – מהד' 2
179	768	CD	The **Java** Tutorial סדנת לימוד
159	586		**JavaScript** סדנת לימוד
199	514		מדריך **ASP.NET MVC 4**
99	824		**ASP.NET 3.5** סדנת לימוד בשפות #C ו-VB
			תכנות
139	288		**Code Complete – מדריך מעשי לפיתוח תוכנה**
169	350		**לחפש באגים**, מדריך מעשי לבודק תוכנה, מהד' 3
99	656		**Visual C# 3.0** סדנת לימוד
139	480		ללמוד **C** - מהד' 3
89	314		שפת **אסמבלי** למחשב האישי, מהד' 2
95	152		יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 4
			PC - חומרה, תוכנה ורשתות
169	428		**מדריך Hacking ואבטחת מידע**, מהד' 2
189	752		מדריך **חומרה ותוכנה** לטכנאי PC - מהד' 5 (כולל חלונות 7/8)
219	608		מדריך **רשתות** לטכנאי PC ולמנהלי רשת - מהד' 4
			Windows
149	544		**Windows 8.1** מדריך למשתמש
19	438		**Windows 8** מדריך למשתמש
19	272		**Windows 7** צעד-אחר-צעד
			LINUX
189	226		**LINUX למתקדמים**, טיפים, טריקים ותכנות ב-BASH חדש

* מחיר מומלץ לצרכן כולל מע"מ

היכנס לאתר להתעדכן בספרים החדשים ובמחירי המבצע בהוצאה

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il

	כולל	עמ'	מחיר*
גרפיקה			
Flash – ספר הדרכה ותרגילים		122	64
אינדיזיין – ספר הדרכה ותרגילים		132	72
Illusatrator – ספר הדרכה ותרגילים		120	64
Photoshop צעד אחר צעד (צבע מלא, למתחילים), מהד' 3 כריכה קשה		200	159
Photoshop צעד אחר צעד (ש/ל, למתחילים), מהד' 3		200	89
מדריך לתוכנת העיצוב והאנימציה **3ds max** (2 כרכים)	CD	1400	289
OFFICE			
יישומי סטטיסטיקה בגיליון אלקטרוני Excel		86	69
סטטיסטיקה יישומית		150	97
טבלאות ציר – ניתוח נתונים חכם		116	87
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 4		152	95
Access 2016 צעד אחר צעד חדש		384	169
Word 2016 צעד אחר צעד		150	59
Excel 2016 צעד אחר צעד		336	129
PowerPoint 2010 צעד אחר צעד		202	69
Outlook 2010 צעד אחר צעד		190	69
Access 2010 סדנת לימוד		360	179
עוד ספרים בגרסאות קודמות (2007 ו-2003) ניתן למצוא באתר הוד-עמי			
ניהול, כלכלה ושונות			
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3		350	169
ניהול ממוקד לעשות יותר עם מה שיש (כריכה קשה) - מהד' 4		358	133
לי זה עולה יותר (תמחיר) (כריכה קשה) - מהד' 3		368	129
מערכות מידע			
Oracle SQL יכולות מתקדמות		626	249
SAS (**Statistical Analysis System**) – ספר לימוד		256	169
בסיסי נתונים ושפת **SQL** – עקרונות ועיצוב		648	149
ניתוח מערכות מידע כולל מתודולוגיית ה- **UML**		818	229
המדריך העברי השלם **UML**		346	329
ספרים דיגיטליים			
לרכישת ספרים לצפייה בפורמט PDF היכנס לאתר לקטגוריה "ספרים דיגיטליים"			
קבצי תרגול לספרים			
קבצי תרגול לספרים שונים תמצא באתר בקטגוריה "קבצי תרגול לספרים"			

* מחיר מומלץ לצרכן כולל מע"מ. קטלוג 1/2017

היכנס לאתר להתעדכן בספרים החדשים ובמחירי המבצע בהוצאה

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il